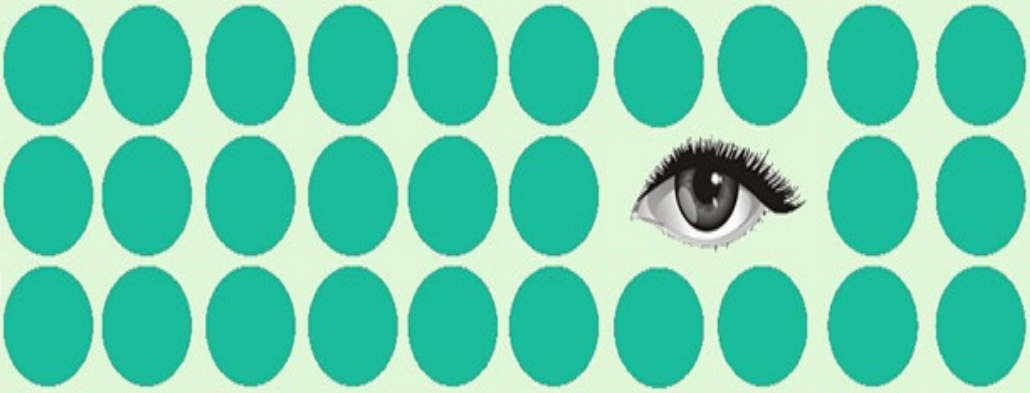
个性化你的阅读



编程狂人

Programming Madman

NO.12

推酷

关于推酷

推酷是专注于 IT 圈的个性化阅读社区。我们利用智能算法，从海量文章资讯中挖掘出高质量的内容，并通过分析用户的阅读偏好，准实时推荐给你最感兴趣的内容。我们推荐的内容包含科技、创业、设计、技术、营销等多方面内容，满足你日常的专业阅读需要。我们针对 IT 人还做了个活动频道，它聚合了 IT 圈最新最全的线上线下活动，使 IT 人能更方便地找到感兴趣的活动信息。

关于周刊

推酷周刊是专为 IT 人打造的行业技术周刊，目前推出的《编程狂人》是献给 广大的程序员们。 我们利用技术挖掘出那些高质量的文章，并通过人工加以筛 选整理出来。 每期的周刊一般会周一的某个时间点发布。

联系我们

tuicool2012　　164644910　　推酷网

下载 APP

Android版本





iPhone版本





2014/02/17/第十二期

目录

封面

业界新闻

斯坦福大学发布免费在线文本分析工具 1

Parse 发布 Bolts，一个面向 iOS 和 Android 的底层库集合 2

9 款最好的基于 Rails 的 CMS 内容管理系统 4

开源世界应该致谢的五家公司 Square、LinkedIn、Google... 8

游戏公司的那些奇葩事 11

前端开发

跨域方法汇总 23

JavaScript 多文件下载 26

写一个更好的 Javascript DOM 库 33

Hybrid APP 开发者一定不要错过的框架和工具 45

编程语言

Python 超级程序员使用的开发工具 47

基于 Capistrano 工具的 Rails 程序部署方案 51

Java IO vs NIO vs AIO vs 协议 Servlet 3.0 以及 NIO 的框架 56

Integer.valueOf(String) 方法之惑 60

在运行期通过反射了解 JVM 内部机制 63

程序设计

Objective-C 相关 Category 的收集 67

豆瓣 CODE 两年历程回顾：git 不是万能的，没有 review 是万万不能的 73
Android WebKit 消息处理 80
Tim Bray：2014 年软件之路 82

后端架构

MongoDB 与内存 87
《淘宝技术这十年》读书笔记 92
探索 Hibernate 新 TableGenerator 机制 97
服务好“最后一公里”，高效 CDN 架构经验 110
探索推荐引擎内部的秘密 116
一起 select 引起的崩溃 130
课堂上传纸条如何防范中间人攻击？ 132

程序人生

《9Tech 访谈录》揭秘游戏王子白泽的技术之路 134
我的北漂在路上--------时不时的停下脚步思考 138
纸书的秘密 145
一个技术人的知识管理方法论 148
ReactiveCocoa - iOS 开发的新框架 157

正文

[业界新闻]

斯坦福大学发布免费在线文本分析工具

斯坦福计算机科学家开发了 etcML（Easy Text Classification with Machine Learning）网站，该网站通过机器学习引擎进行文本分析，目前该网站向所有人免费开放。

斯坦福大学官网对此进行了说明：etcML 网站基于成熟的文本分析引擎，估计总体的情感倾向（是消极还是积极）。etcML 项目负责人 Richard Socher 表示：

“我们想开发一套供普通人和研究人员使用的标准的机器学习技术，所有使用者都不需要编程能力。”

斯坦福大学候选博士研究生 Rebecca Weiss 表示：

etcML 提供了一种简单的方式，对单词和短语所体现的观点进行分类。

我可以训练分类器，也可以对任何一段文本打上标签，实现这一切并不需要写一行代码。我还可以和记者或研究人员分享我的分类器。

斯坦福大学计算机语言学研究者 Rob Voigt 通过 etcML 对 Kickstarter（一个创意方案众筹平台）上的项目进行评估。他发现，多使用“我们”、“我们的”这些复数代词能够提高项目宣讲成功的概率。Voigt 表示：

“我们并没有声称我们的分析是决定性的，etcML 分类范式提供了有价值的成功线索。”

斯坦福大学计算机科学博士生 Chinmay Kulkarni 使用 etcML 对 2000 名学生的简答题测验进行打分。此前，简答题打分是通过学生之间的互相打分完成的，平局每道题有 4 个学生进行打分。通过 etcML 打分后，平局每道题只需要 3 个或更少的学生打分。Kulkarni 表示：

“我们获得了同样的精确度，同时减少了学生的负担。”

Kulkarni 发布了这一项目的论文。

Socher 认为 etcML 能吸引许多人尝试有趣的语义分析项目并提供反馈，从而

提升 etcML 背后的计算引擎。

值得一提的是，Andrew Ng（吴恩达）是 etcML 项目的顾问，他是斯坦福人工智能实验室的主管，也是在线公开课程平台 Coursera 的创始人之一。

原文链接：http://www.infoq.com/cn/news/2014/02/stanford-free-text-analysis-tool

Parse 发布 Bolts，一个面向 iOS 和 Android 的底层库集合

数月前，Parse 被 Facebook 收购。最近，它开源了一个面向 iOS 和 Android 的底层库集合，统称为 Bolts。根据 Parse 的公告，Bolts 是 Parse 和 Facebook 共同努力将两家公司各自独立开发的小型底层工具类合并的结果。

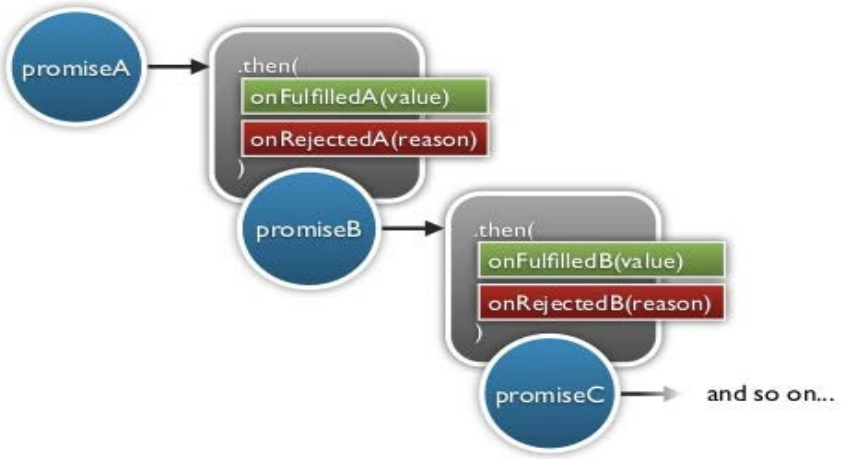
Tasks 是 GitHub 上第一个可用的 Bolts 组件，旨在按照 JavaScript Promises 模型处理异步操作。

Promises 试图解决使用回调函数处理异步操作时通常会出现的若干问题，尤其是这样一个事实：由于回调函数内部异步操作的嵌套，试图组合多个串行或并行异步操作会很快变得难以处理。

为了这个目标，一个 Promise 代表一项可能已经完成或者可能尚未完成的任务的结果，而它最终可能会变成一个错误。这样，任何异步操作都可以立即在执行结果中返回一个 Promise；该 Promise 可以随时访问，如果异步操作尚未完成，可能阻塞调用者。

不过，一个 Promise 通常关联两个回调函数，用于在异步任务已经完成或者失败时调用。Promises 的特别之处在于回调函数本身封装在 Promise 之中，所以它们只在将来的某个时间点执行，或者根本不执行，这依赖于原 Promise 的状况。

```
var promiseC = async()
                .then(onFulfilledA, onRejectedA)
                .then(onFulfilledB, onRejectedB);
```



多亏这一机制，处理异步操作的序列变得简单易懂，因为 Promises 可以链到一起来代表异步操作和其回调函数，如上图所示（源自：Promises，Luke Smith。）

Promises 的另一项优点在于错误通过 Promises 链传播的方式：由于 Promise 知道它是否已经达成，它可以将错误状态沿着 Promises 链传播，直至找到一个错误处理器，因此，开发人员无需为链上的每个异步操作提供错误处理器。

Promises 实现的组件可以用在 JavaScript、Scala、Clojure 和许多其它语言中。

Parse 声称，与 Android AsyncTask 和 iOS NSOperation 相比，Tasks 有若干优势，其中包括：

连续执行数个任务不会像只使用回调函数时那样创建嵌套的“金字塔（pyramid）”代码。

Tasks 是完全可组合的，允许开发人员执行分支、并行和复杂的错误处理。

开发人员可以按照执行顺序安排基于任务的代码，而不必将逻辑分解到分散的回调函数中。

Bolts 组件与 Parse 或者 Facebook 服务完全无关，不需要使用 Parse 或者 Facebook 的开发人员账户。

尽管已经发布了更多的 Bolts 组件，但 Parse 尚未发布与之相关的任何细节。

原文链接：http://www.infoq.com/cn/news/2014/02/parse-announces-bolts

9 款最好的基于 Rails 的 CMS 内容管理系统

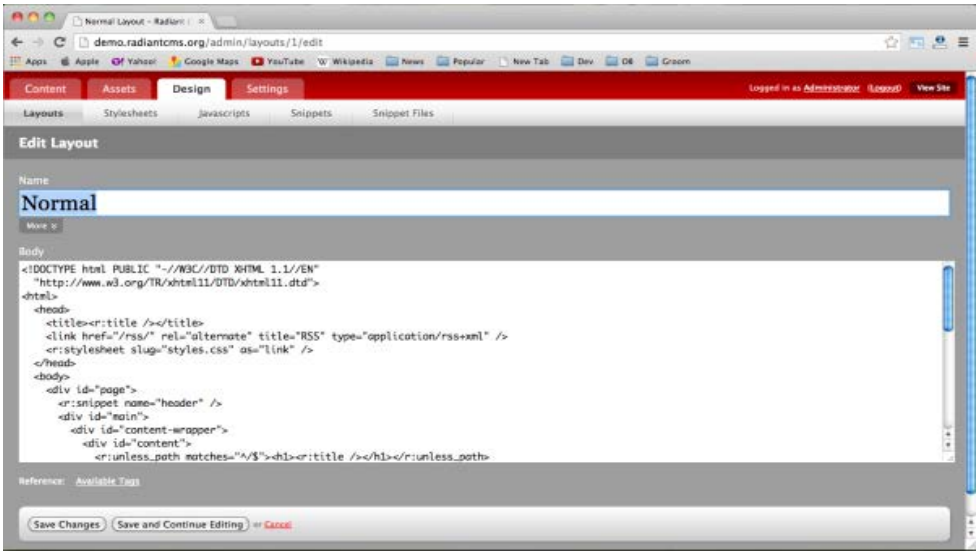
内容管理系统（CMS）是用来创建，管理或者是发布内容的计算机程序，CMS 通常是在网站和博客上使用，多用在公司网站，可以免去处理编码的问题并且提供非常易于使用的用户界面。CMS 变得越来越流行，因为它不要求太多的编程语言知识，实际上是给用户提供一个管理内容的环境。所以，即使是新手也能够利用 CMS 很好的完成相应的工作。很方便的使用 CMS 通常或者是公司网站上使用

当然，现在有很多种语言可以创建 CMS ，而 Ruby On Rails 就是其中一种。Ruby On Rails 是个开源 web 应用框架，用来运行 Ruby 编程语言。这个开源框架主要功能有：运行 web 应用，提供创建网页的工具，收集 web 服务器的信息，查询数据库和处理模版。综上所述，Ruby On Rails 能提供各种强大的功能和安全的用户界面，帮助新手更好的完成他们的工作，不要求他们有太多编程方面的基础。

Ruby on Rails 内容管理系统提供各种功能来帮助用户更好的管理网站，所以推荐给 web 开发者。今天我们要跟大家分享的是 9 款最好的 Ruby On Rails 内容管理系统，帮助大家提高生产力，节省您宝贵的时间。

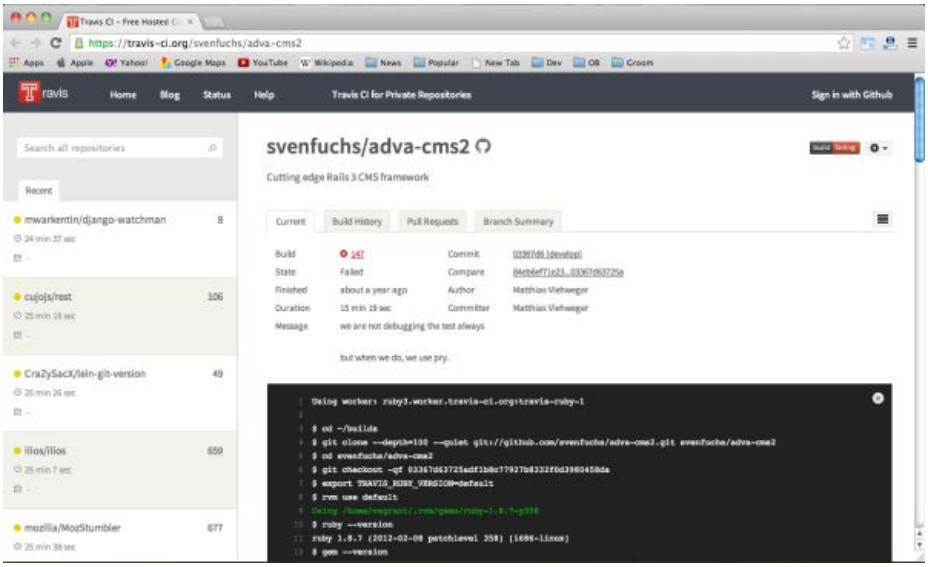
1) 最好的 Ruby On Rails CMS- Radiant CMS

Radiant 是最流行的开源内容管理系统之一，有着许多令人着迷的特性：优雅的用户界面； 一系列有着迷人布局的模版；易用的各种插件系统，等等。



2) 流行的 Ruby On Rails CMS – Adva CMS

Adva-cms 是个引领时尚前沿的基于 Ruby On Ruby 和 Rails 引擎的开源应用平台。



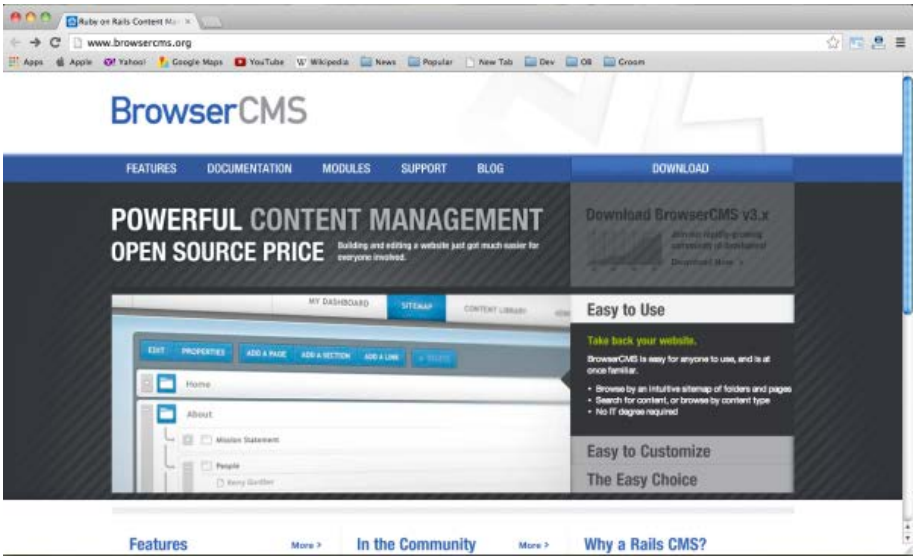
3) 最好的 Ruby On Rails CMS – Casein CMS

Casein 是个开源，轻量级的 Ruby on Rails 内容管理系统，有着极小的数据 CURD 接口，用户身份验证和基础的管理系统。



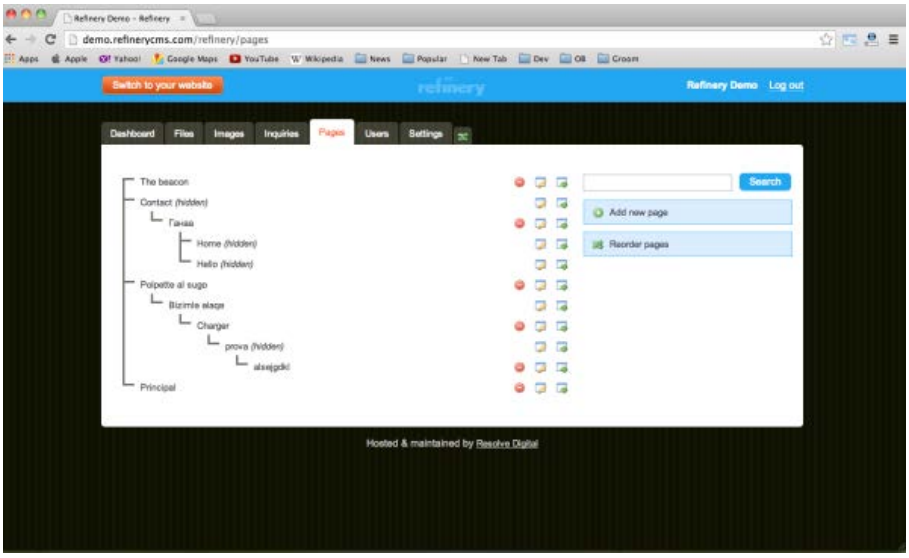
4) 令人印象深刻的 Ruby On Rails CMS – Browser CMS

由于 Browser CMS 有着健壮的功能特性，它成为了众多编辑器中最受欢迎的 Ruby on Rails CMS(内容管理系统)，对每个人来说都非常简单易用。



5) 最棒的 Ruby On Rails CMS – Refinery CMS

Refinery CMS 是 100% 免费，允许用户依据自己的意愿去使用和修改。它给用户提供一个漂亮简洁的界面去创建可管理的自定义内容网站。



6)最流行的 Ruby On rails CMS – Locomotive CMS

Locomotive CMS 倾注于当前最流行的前端技术，标准的开发流程，不需要网站的用户花费任何的时间去学习使用方法。

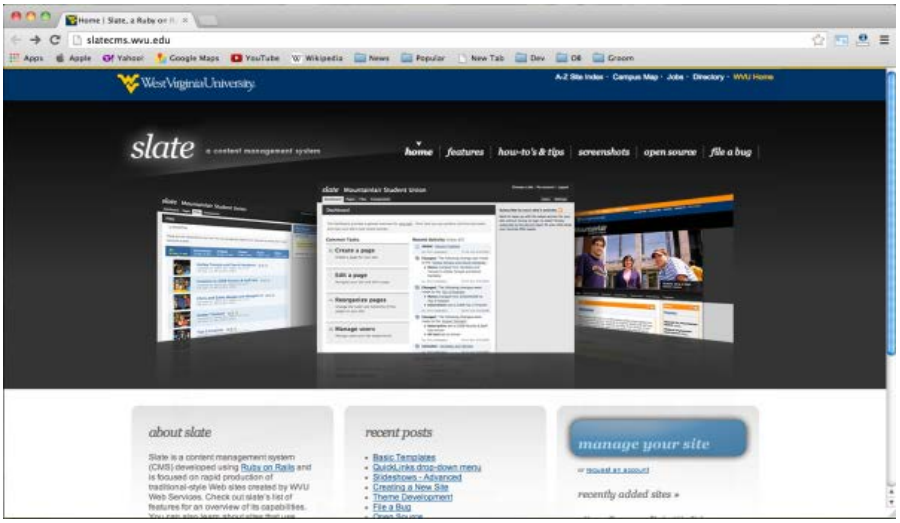
7) 最好的 Ruby On rails CMS – Comfortable Mexican Sofa

ComfortableMexicanSofa 是最权威最流行的基于 Ruby on Rails 3 的 CMS 引擎应用之一。它有着很强大的功能：能很好的使用标签创建网页模版；只需要单个安装就能创建多个网页；等等。



8) Ruby On rails CMS – Slate CMS

Slate 是个 Ruby on Rails 内容管理系统，帮助用户使用 WVU web 服务来创建现代化风格的网站。



9) 流行的 Ruby on rails CMS – Webiva



原文链接：

http://www.oschina.net/news/48760/best-ruby-on-rails-content-management-systems-cms

开源世界应该致谢的五家公司 Square、LinkedIn、Google...

LinkedIn 的工程师们开发了、并至今依然在维护着 Apache Kafka messaging program 项目，不仅在 LinkedIn 内部运用广泛，自从将 Apache Kafka 项目开源和免费授权（使用）后，例如 Twitter、Tumblr、Netflix 和 Pinterest（更多 Apache Kafka 用户）都成为了 Apache Kafka 项目的受益者。

可能有人会感到迷惑，为何巨头公司们会热衷于将代码贡献到开源社区？事实上，任何人都可以成为开源项目的贡献者和受益者，任何人都可以通过例如 GitHub 或者 Apache 软件基金会等途径参与到开源代码的贡献、调试、下载和分享中。



那么，商业公司如何从开源项目中盈利？难道他们不担心会将自己的秘密项目曝光？为何浪费时间去和开源社区打交道，如果只专注于自己的核心赚钱业务难道不是更好吗？

以下为 Readwrite 记者 Lauren Orsini 收集到的一些开源社区参与者的反馈。在她看来，所有的开源参与者都应该感谢以下五家公司给整个开源界带来的贡献。

Square

总雇员数不超过 700 人，移动支付公司 Square 是这份开源贡献名单 Top5 里规模最小的公司了，但给开源社区的贡献是卓越的。

据 Square CTO Bob Lee 的说法，Square 已经将超过 60 个项目提交到开源社

区，贡献了 25 万行左右的代码。

Square 贡献的开源项目中最火的是哪一个？或许是 Picasso，作为 Square 开源的一个用于(Android)系统下载和缓存图片的项目，目前已经广泛应用于 New York Times，Paypay，Ouya，Spotify 和更多应用中。

在 Square CTO Bob Lee 看来，正是因为 Square 公司规模小，因此将代码开源更具积极意义。与其将内部有限的开发资源都投入到通用问题的处理上，不如将其贡献给开源社区并与外部海量的开发者协作（重构和持续发展这些项目）。这种做法同样适用于和 Square 类似的公司，集中精力完善业务上的特性产品，而减少耗费在通用基础架构上的时间。

Google

Google 能够列在开源贡献名单 Top5 里的原因，并不是它所贡献的开源项目数量，而是胜在质量和规模。举例来说，Android 和 Chrome 只是单个的开源项目，但围绕它们构建了庞大的生态圈，基于它们（平台）上所开发的插件和应用数量惊人。

Square CTO Bob Lee 同样认为，Google 的开源平台不仅可以引发出一波开源项目，还可以带动该开源项目的持续发展，例如 Square Dagger 新库，它就是一种针对 Android 和 Java 的快速依赖注入器，该库的源码已经发布在 GitHub 上。目前 Google 对 Dagger 的贡献甚至已经超过了 Square，现在 Dagger 也已经是一款免费的软件。

对于长期致力于疯狂赚钱的 Google 来说，投身于开源社区的主要诉求并不是索取回报，而在于社区建设本身。 Google Android 开源项目经理 Christopher Katsaros 表示。例如 Google’s Summer of Code，就是 Google 主办，旨在激励大学生为开源项目做出贡献的项目，目前已经为开源社区贡献了 5 千万行代码。

LinkedIn

多谢 LinkedIn 开源 Kafka 这样的高吞吐量分布式消息系统，目前，LinkedIn 已经通过 80 余个不同的项目为开源社区贡献了超过 50 万行的代码。是什么推动了 LinkedIn 持续投身开源，LinkedIn 首席工程师 Jay Kreps 认为，这将有助于产品保持高标准并引发人们的关注。“要鼓励优秀，你必须公开化！”Jay Kreps 认为，工程师和其他人群一样，如果每个人都在关注（我们的产品），我们就必须让它

看起来很优秀。LinkedIn 所用的所有内部工具都将开放出来给任何人使用，只要它不和公司的商业计划直接相关，任何人都可以免费读到这些项目的代码。让任何事情都成为秘密武器并不是我们热衷的有效战略，Kreps 补充到，内部的（工具和技术）信息并不需要成为我们的竞争优势。

相反，坚持开源策略是最好的技术招聘广告。开源能够在一定程度上 LinkedIn 是一家优秀公司，外部的人能看到我们的每一行代码并参与贡献。很多 LinkedIn 招聘到的人都说这是他们加盟的重要原因。

Facebook

My ReadWrite 作者 Matt Asay 此前有文章认为，Facebook 已经成为全球最大的开源公司，甚至超越了传统的 Linux 界的 RedHat。

从创业时起，Facebook 从头到尾都是构建在开源（技术）基础上，离开开源就没有 Facebook。软件工程师 Jordan Delong 在 Facebook 开源 Folly 标签库后撰文给出了这样的结论。Folly 是 Facebook 的一个开源 C++11 组件库，它提供了类似 Boost 库和 STL 的功能，包括散列、字符串、向量、内存分配、位处理等，用于满足大规模高性能的需求。这也是 Facebook 为何通过开源各种工具并积极回馈开源社区的原因。令 Facebook 在开源贡献榜脱颖而出的另一个原因是，它是唯一一家将整个数据中心全开源的公司。Open Compute 是一个庞大的开源工程，它使得数据中心的每一个组件完全透明化，从供电系统到服务器。

Facebook 还是在开源界贡献持续时间最长的公司，与其他大量仅在开源界活跃三四年甚至更短的公司相比，Facebook 从 2006 年就开始谈论开源精神并坚持至今，即使不论 Facebook 在开源界贡献的代码数量，它的这份坚持精神都令其他公司望其项背。

Twitter

Twitter 在开源社区的贡献是显著的，甚至其贡献的某些开源项目在一定程度上比 Twitter 本身带来的影响力更大。用 Bootstrap 举例，它起始于 Twitter 内部的一个小工具，最初的目的是让它看起来更漂亮。但自从 Twitter 在两年半前将 Bootstarp 项目贡献给开源社区后，它现在已经到达了一个新的高度。目前有成千上网的开发者利用 Built with Bootstrap 和 Bootstrap Expo 创建应用。

随着 Bootstarp 3.0 的发布，Twitter 开源负责人 Chris Aniszczyk 宣布将其完全贡献

给开源社区。Bootstrap 已经不再是 Twitter 控制下，大量的贡献并非来自 Twitter 自身。Chris Aniszczyk 说看到一个开源项目的腾飞是如此令人兴奋，但他也坦言 Twitter 坚持开源策略的另一个目的是为了“掌控未来”，这也是为什么 Twitter 最近向开源社区推出 CocoaSPDY 的原因，CocoaSPDY 作为针对 iOS 和 Mac OS X 平台的 SPDY 开发包，可加快 HTTP 请求。

通过致力于 SPDY 的努力，能推进整个工业界的方向。人们将支持 SPDY，使得网络更快，使 Twitter 的用户体验更快。这不仅仅只是 Twitter 的战略方向，而是整个业界推进的方向，最终目的都是为了提高用户体验。每家公司为开源做贡献或许都有其诱因。但通过 CocaSPDY 这样的项目，最终对于用户的结果是一样的。它将使得在线用户体验更好。

原文链接：http://www.csdn.net/article/2014-02-10/2818351-open-source-5-companies-code-projects

游戏公司的那些奇葩事

不知道是我这人记忆力特别好，还是运气比较好，貌似见过的奇葩比一般的人都要多一点。

先挑几个爆爆

男女关系篇：

1、某游戏老板跟公司内某女员工拍拖许久，最后搞出个小孩出来

不结婚，扬言说女人你要么留下带小孩，要么滚蛋后来这老板把这个小孩的户口上到自己弟弟的户口上

2、某游戏公司老板跟某女员工拍拖，后此女被甩，公司内某屌丝男员工接盘，还总是觉得这家事情很自豪的样子，自己女人是土豪老板睡过的，现在属于他，好像沾了多大喜气一样，我其实一直觉得他不太正常……

3、某游戏公司研发部门某男，属于……真有点不清白的那种类型，只要是个女的就去贴，不管结婚不结婚，有无男友，连打听工作都不去做，直接就冲过

去示好，答主身边的很多屌丝程序猿在公司内找的女朋友都被此男贴过，很多人其实都想揍他，这还是小事，关键是，他自己把隔离工作都做得很好，能同时跟几个不同的女人保持关系，搞大了一个肚子，打掉，还没24小时就去跟新入职的女同事主动送早餐，在同一家公司出现过上班时间，两次，为情扭打的情况，游戏圈常关注各大 QQ 群的小伙伴们可能也应该知道一点，反正，极其混乱。

4、有一个极品男，特点是总是穿山寨奢侈品在外露脸，这事儿貌似有人捅出来过，就是那种什么事情都打肿脸充胖子时候还死不承认的尿性，花女朋友的钱，刷她的卡，后来那女人四处往游戏圈发消息找他闹，要他还钱……

5、某公司老板，招了一个美女做助理(其实也一般般而已，但是在 IT 圈子里还算行)第二天，美女的诺基亚手机变成 Iphone，第三天，美女除了台式机以外又多了一台 Thinkpad，手机 APP 项目组的同事愤愤不平得在 RTX 里说，尼玛我们申请个 IPHONE 和 IPAD 做测试机几个月都不批!过几天，说这个话的人消失了，再过段时间，这个美女桌子上多了一台 AIR，到处求人想把系统换成 WIN7 没人理她，最后是公司近40岁的一大叔帮他弄了

职场政治篇：

1、86年出生的小屁孩，没什么实质工作经验，1年内在上海连续跳槽数次，好几家公司都呆不下去了，后来有一点跑路的性质，去了广州，装经验丰富的制作人，开价20K，居然被收了，恩这好像是2010年时候的事情，现在游戏圈子里面很多在这家公司呆过的人，说起他都是一脸厌恶，据说有人组织了兄弟想去挑他手筋，后来还是忍住了，没闹起来。

2、你知道是什么公司收的么，那时候是个新成立还不满1年的某运营平台这个平台前不久卖个某通讯公司，这公司的老板做生意也是比较有忽悠性质的，不算是有眼光，运气好误打误撞，接了一个 A 级产品的联运，就这么做起来了这老板在跟上面那个人网聊的时候，炫耀自己公司的业绩，把后台的数据截图给他看后台显示的数据单位其实是元宝，而不是 RMB，做游戏的都知道，后台显示100W 元宝的收入，其实只相当于10WRMB，1比10的比率嘛，但是上面那个案例中的人理解成了 RMB，屁颠屁颠打飞的跑去广州见面，事后居然还跟周围同事说起这事，愤愤不平地说自己被骗了……被骗了……被骗了……

3、某运营平台下面开设的某研发项目，(好吧，我删掉部分文字否则太容易

猜)这个项目有四位主策，同时在职，居然还可以和平共处，他们在招聘每一个策划的时候，都给人家挂上主策的帽子，然后抓进去用几个月，觉得用得差不多了就再去招一个，恩，主策划原来这么不值钱，随便谁都可以来做，也随便都可以换掉。

4、人家都说，通常在高管中，拥有心理疾病的人占比较高，我刚开始工作不久，其经历就证实了这一点，这个是好多年前的事情了，那时候单反还不像现在这么普及，我有玩单反的习惯，因为我父亲以前就是摄影师，他平时总在给我灌输，后来我参加工作，小新人一个，公司活动，我心想这么难得的机会，我就带着自己的设备去给活动拍照，本来是个好心，这都没什么，有一天晚上加班，我想找个安静的地方，就拿着自己的笔记本去了会议室，一墙之隔就是某高管的办公室，隔音效果不太好，我清清楚楚得听到他点名在问 HR，我是怎么招进去的，工资是开的多少钱之类的，是不是工资给的很高，几分钟后我打算再去换个安静点的地方，走出会议室撞到他，他居然来一句”你用笔记本?“我说”恩啊，XXX 您有什么吩咐么“他没回答，扭头就走了，第二天，他拿着一张离职表过来，丢到我位置上，这世上真有这么见不得人好的么……

5、这个是听来的，好多年前。那时候这某公司还很小，人不多。综合部门的头儿，女的，统管 HR 行政采购。奇葩之处在于，她很喜欢组织活动，一般都是一些团购，周末旅游，外出聚餐之类的事情，很多人想，经常有活动是好事啊。

但问题是，每周都不消停!而且不是公司出钱，都要 AA，而且她自己通常都会带老公去，然后自己不用付自己的那一份钱，让老公出这一份

后来听一个行政的小妹说的，她有一个奇特的行为，就是每个月发完工资后，她会将工资表和自己做的未来一个月的活动安排表拿出来排期，哪些人几号参加哪个活动，以此类推，据猜测，基本上她的安排是以把这个人的工资用得差不多为止，等表格安排完成后，接下来就开始一个一个邀约，不去的人通常都会收到过他的白眼和背地里的闲言碎语，每一次活动还必须要发票。这个透漏的小妹离职前跟我们说，他看到每个月光发票报销就两三万。

6、有某游戏媒体公司，也想参与研发，就在本部外面租了一片商住楼，规模不大，总共也才20来人的样子，打包收的一个团队，先试试水嘛。

安插了一个“信得过”的非技术非项目出身的人过去做负责人，其实就是个

会计。

谁都看得出来，他一直都想扩大自己在公司的控制力，所有的行政财务人事的事情他全包，连晚上加班定个外卖都非要亲自经手，你想事后拿着发票找他报销是绝对不会给的。

但是他不是程序员出身，一直苦于不知道如何进行项目管理，总是很明确的表示想要介入，研发的人本来就是打包的，谁会理会你啊。

每周周末就会有这样一个画面，一堆研发的人围着这个会计，给他做每周工作汇报，其实谁都知道他听不懂，但是每个人也慢条斯理的说上10来分钟，这大半个下午就这么混过去了。

7、某研发团队，曾经用 PHP 语言做过一个项目，有小成也谈不上什么非常好。

后来决定再开项目，其实立项工作还是做的不错的，目标明确，人员也比较齐，应该用不了多久就可以做好。平台合作方就选择的某某某某，某某某某注资。

突然一下老板出面，说 PHP 这种语言不好，我们应该换成 erlang，理由是运营合作方的要求，说以后的程序都应该使用这种新型语言，因为单台服务器的承载能力比较高，可以减少运营成本。

然后全员停工恶补 erlang……

混过广州圈子的朋友们可能猜得到，这是某某某某平台的量产策略所导致，因为这个平台曾经有一款使用 erlang 语言的成功产品，他们就想去复制这种成功，把源代码四处分发，打着协助的旗号，用天降神兵的姿态，游说旗下各工作室和有合作关系工作室均按照他们的思路来出产品。

这个研发团队的老板估计是眼界不够，听信忽悠，就做了个这么的愚蠢的决策。

看到这种局面，不知道各位会作何感想?连你用的语言都可以这么随随便便让运营方给改了?

公司里面有一个很资深的 C 程序，算是很核心的技术人员，他原本的主张是想亲自带公司这些还不够成熟的小程序员们，慢慢转到“正轨”上来，但是因为这一件事情，上面对他的态度有一些微妙的变化

并不是他抗拒 erlang 这种语言，他是团队里面最快过渡到新语言的人。而

是这个老板又不知道从哪里听信流言(这一点并未确认是这个资深程序说的),说C++价格贵人又屌不适合长期使用,建议多使用新人,反正现在的 IT 培训机构和高校产出了大量面对对象的程序员,省成本

整个团队磕磕碰碰,终于用新语言把第一版本 DEMO 交付之后,这个老资深缺消失了

后来各种改,导,调,改,导,调……各种不稳定,到了 Deadline,项目依旧是不稳定。

后来这个团队就整个被某某某某收了,连办公场地都搬到人家那里去了,老板降级变成了项目经理,又没过多久,团队给拆了。

这一条可能跟公司内部的职场政治没多大关系,但是,公司与公司之间也是有政治的……

8、再说一个公司与公司之间的政治

之前某某某某平台还没涉足研发的时候,通常都是跟一些小研发团队保持合作关系。

有一个人 A 跑去跟某平台沟通,然后某平台统一注资。

筹办公司、租场地简单装修、招聘、挖人、开搞,一年多以后,项目出来了,虽然还不够完美,但是起码有个样子,A 很天真的交付给平台。

但是对方一直都是不紧不慢的,量也不导,总是找各种细节问题让他去改,一连改了好几个月。

然后这个平台的自研项目上线,类型题材风格均类似,成为该平台主打产品。

几个月后团队动荡,被平台挖角。

后来 A 出局,不知道又去干什么了,失去联系。

为人处世篇:

1、在一次中间件服务商组织的客户酬谢酒会中,跟广州某运营平台的 CEO,打过一次斗地主

这个平台不久前卖给了某传统公司

这家伙真的……牌品奇差,他还没达到半个小时连输了几把,极其痛苦的表情和情绪

也不说话,就这样憋着,憋着……

后来我实在觉得气氛太紧张了，我就说，我们仨交换下名片吧，正好休息一下，要不要再叫杯喝的?

这家伙居然直接弃桌跑去打台球，走也不打个招呼，把我和另外一个 MM 丢那不管了……

牌品即人品，这句话一点都不假，在后来的合作中完全可以体现……

好吧，这家伙名片是蓝色的

2、某公司的 XX 副总裁，有时候在外面也自称是 CEO。

在曾经不是做高管的时候就喜欢出去做点泡泡妞之类的事情，也总是调戏调戏公司内女员工。

算是个猥琐男。

后来居然有小姐找到公司来闹了……

后来大老板肯定是说希望不要再出现这种事情之类的话啊

接下来

第二次发生闹事的时候，人家小姐指着他们两个人的鼻子 BLAH BLAH BLAH

再后来，公司换地址

我不确定这公司换地址跟着有没有关系，反正大家都这么说

3、好几年前的一款页游，算是国内最早的一批 ARPG 成功项目

研发成本不足百万，研发周期不足半年

谈下来的第一家联运平台月实际收入就超过了600W+，对外宣称1500W+

研发人员月奖金，最高一个月420块钱……

奇葩么?看官们能感受到共鸣么?后来陆陆续续人都走了

这项目现在2年多没实质更新，仅仅只是运营偶尔做做活动，换一下宝箱内的道具而已

现在月收入还能保持在200W 左右，后面推出的很多 ARPG 都挂了，但是它还活着……

我跟这个项目组里面很多人的关系都挺好，人散了以后，一直都有联系

奇葩的事情在于，我现在收到的很多简历，很多没见过名字的，都写着在这个项目有过经验

有个家伙 A，就靠着简历里面的这一条虚假经历，成功混到了某手机游戏公

司。

A 进去之后，约了老公司里面的单身女同事出来吃400多一位的自助餐，各种吹嘘。

然后这个女同事把这事情讲给我听了。

A 做了某管理职位一年，期间开了一个手游项目，管又没能力管，最后跑路，项目黄了。

4、某创业公司老板的观念本来很好，要高薪养廉，要充分信任人和调动人积极性。

但是他貌似没意识到，这里是中国大陆，并不是新加坡。

月薪30K 请来一个大神级主策划，名字是。。。两个字

据说每天会把脚翘到对面老板的台式机机箱上打瞌睡

据说会把自己做的 EXCEL 表格完全加密，单元格上面的公式、函数、数据引用都看不到

后来连老板都怒了，跑去要求他解开表格密码

最后把老板的钱都烧完了，项目还是没搞出来。

这位大神去了一家手游公司，就是上面那个案例末尾最后说到的那一家手游公司，正好接了上面家伙 A 的位置。

5、某研发工作室的老板，貌似手上有好几个公司的空壳，但是大伙儿都不知道。

在公司 A 名下，组织研发团队，搞出产品来之后，许久也不见上线。

给兄弟们说，公司要解散，项目黄了，大家珍惜珍重吃饭喝酒拍肩膀祝福散伙。

后来在某页游平台某项目火了，有人开始质疑为什么看起来这么眼熟?

有一个前端程序反编译后发现，这 TMD 就是老纸当初写的，不是说挂了么!?

后来一调查，这个项目的负责人就是同一个老板，但是是挂在另外一个公司的壳下面的。

近年来，我发现越来越多的人善于使用这类手法了……

6、再说个虚假简历的事情

也是业内某平台，在运营出一点成绩之后就想发展自研团队，当时从上海招

了一个大神级角色，也是制作人身份进来，这家伙算是不务正业型的，在公司2年多时间内：

接手一个因主策划老家出事而请辞回家的项目，不到半年，本来好好的一个项目黄了

后向老板建议开设一个仿 FACEBOOK 某休闲社交游戏，高调组建团队但是又不在项目里亲自起作用，唯一干过的事情也初期给老板建议开项目，半年后，项目出来，太粗糙，团队解散

接下来又向老板建议要进军手游行业，再新开一个 IOS 宠物游戏，有点像“会说话的汤姆猫”那种，半年后，项目黄了

组织玩得好的同事们，一起拍所谓的有关梦想的视频，还在公司里面得意得流传。

接触到一种新的游戏开发引擎，是国外开源的，在网上收集资料再加上自行脑补 PPT 出来，跟人力资源部门合作组织“培训课程”，然后去给公司的其他程序员“讲解”，大家私下评论是“这 TMD 都是网上的东西还培训个 DICK!”

后来结识了一个公司外的某研发组 A 的人，把他们的项目 B 介绍给自己平台做运营，跑去运营部对着运营专员指手画脚，这项目要怎么做怎么做要让研发他们怎么该怎么改。

后来又建议把公司的一款老页游改编手游，跑去找公司另外一个手游部门的同事，人家不鸟他，后来这家伙居然自己单独又组织个“手游部门”开设这个项目(所有人都不明白老板是怎么允许公司内存在同职能部门的)，最终这个项目的结局也不用我说了。

在距离公司年会还有2个月的时候，上班时间，擅自占用公司大会议室在里面手舞足蹈，说要排练舞蹈节目，搞得需要用投影的同事们怨气很大。

基本上没有一件事情干到最后。

一段时间后，上面所说的某研发组 A 的同事，有一天拿着一份简历跑来问我们，这人到底是谁?

简历上面基本上把这个平台的大部分自研项目都纳入自己简历名下，又是讲师、又是导演、其中还有一条就是项目 B 的首席制作人头衔

简历注水的人不少，但是这个是我见过最…那个的，哥们儿你也太能灌

了……

7、某规模较大的页游平台，据说在公司内组织神马德州扑克竞赛

这事儿在这里写出来没效果，但是如果你是一个熟悉德州扑克游戏的人，在场观察这个活动估计会笑死。

基本规则是错的，在一轮内，加注不限次数，在盲注后我加到100，你加注到200，我加到400，你加到600，我还能继续加到1000，请问各位看官，你 TMD 有见过这么玩的?

筹码汇率中途改变，刚开始的时候100RMB 可以从银行购买100筹码，后来筹码池处于弱势的那个人(其实就是这个赛事的提议者)建议改变汇率，100RMB 可以从银行购买1000筹码，众人鸦雀无声，然后他继续说，好了，给我来100块钱的。改变一次也忍了，后来越改越高越改越高，后来桌子上每个人桌上好几十万都花不完，比赛因此陷入僵局，谁都干不掉谁……

最后局面僵持不下，围观群众众多，场面上三个人分不出胜负，直到上面说到的某人终于有一局拿到三条 A，然后他居然厚着脸皮建议说打的太累了，不想打了，我们这一局定胜负好啦，大家都 All in，不愿意 All in 的就算输，牌友们实在是坚持不下去了就都同意了，结果就是这人赢了。

后来还在公司内开会，这人还专门准备了资料和演讲，说自己是怎么运用概率学、心理战、演技获得这个德州扑克冠军，都可以扯到他学生时代是怎么打牌的。

这事儿在接下来一个月的饭局内都是几个德州牌友间的重要笑点

你说现在这公司都是招的些什么人……

8、在页游早期，没有 SP 背景的人，刚开始其实不懂得如何做数据分析，很多项目人都不知道到底赚不赚钱，只能通过口头沟通了解运营情况，反正那年代也没个什么管理后台之类的东西。

某项目年底上线，口头流传说只是做平一般般而已，谈不上赚钱

一个月后，临近春节放假，发过节费(事后大家互相一对数量，均在200-500不等)

下班前，老板拿着写着名字的信封一个一个给，最后总结发言，说我们多惨多惨，公司账面上仅仅只有几万块钱什么什么的话，但是还是不能亏了大家什么

什么的，这次发奖金只照顾基层员工，管理层一律不以享受等等的话，然后很悲情的下班了。

但是(终于出现这个词了，我知道你们在等)

当天晚上加班到12点钟还没走的同事们，看到老板带领着好几个高管回到公司，还有几个不认识的人，都喝得红光满面的，各个手上不是新出的手机、平板、笔记本之类的一堆新电子产品

第二天上班，下班，过了一天。

第三天，传闻老板没出现在公司的原因是：在某会所被抓了

9、某平台 A，老板据说是个年轻富二代。

很多人都说他有一个很令人难受的特征就是很喜欢不懂装懂。

有一次我陪某项目组的头儿，拿着产品去跟他谈联运，这个产品已经在另外一个平台已经上了，虽然说数据不太好，但是也不至于太难看，再说产品也有很大的改进空间。

我们演示完之后，就比较谦虚地说，我们的产品其实还不完美，我们打算如何如何去调整，反正单方面说了快一个小时。

他的反应是：“这不行啊，这项目还不好啊，要改啊!”

我有点汗颜，后来这朋友继续说，我们已经在某平台上了，头半个月的数据是这样的，虽然还不够好，但是我们有信心……“之类的话

他跑去把数据一看：“这不行啊，这数据不好啊，这要提升啊!”

我瀑布汗，我说我们会改好的，我们会先把美术的哪里哪里哪里给调整了

他的反应是：“这美术不行啊，要更漂亮才行啊!你们的美术不行啊!”

我大概知道他的套路了

后来吃饭、其他活动，我就尝试问一下他过去是做什么的，后来我跟他聊他的车的牌子和型号什么什么的

过了没多久貌似连车都换了……

我真不是故意刺激你的……

10、有人说，一切历史都是近代史，这句话我现在要举起四肢表示同意

某公司老板 C 的历史：

未读大学，来南方打工，C 期初的工作就是在游戏平台做运营专员

那个时候其实页游还没兴起，他运营的一个项目是内地开发商做的一个端游项目，反正没做起来，后来也搞了一些小游戏什么的，这都不是重点。

后来出现了页游这种新兴事物。

当时这个平台的资方和高管，有派系的，主要的派系一直都比较清高，次要的派系都比较“聪明”。无论是主要还是次要的派系，他们都有着上市的目标。

当时大家都觉得这种新兴事物有市场，决定要进军页游市场，当时次要派系的头儿 B 就自己单独又筹办了一家公司，拉上 A 注资，让 C 过去做了名义上的老总，占一小部分股，然后很多业务上的事情就直接照抄原平台主要派系的做法。

后来这个单飞的公司有起色了，还留在老公司的人总是耿耿于怀。

招聘和新人篇：

1、某公司某次搞大型招聘会，要跑好几个学校。

提前1个多月就开始准备，号称给应届生的最高年薪可以达到12W，公司官网上都挂上灯片广告了

后来我们这几个抗拒不够激烈的人，迫于 HR 部门的攻势和贴身游说，被抓去当面试官

其实面试的过程也还 OK 啦，每天有很多青涩活力的面孔经过，有好多还很主动的要我留联系方式，经常还会在 QQ 上问一些问题

经过筛选后确定了一批人选，后来请到公司来专门为这些准应届生开动员会，培训会，签三方协议，搞得很隆重的样子，后来他们忍受着极低的每日补贴干了好几个月，好几个真的很卖力

要做毕业设计，回校，再来公司，将三方改成正式合同，结果只留下来三个，最高的一个薪酬签了月薪4500，结果我们这些面试官成了忽悠，好像是我去年给他们承诺的高薪不予兑现，花无数时间去跟他们一一解释

第二年同样的招聘活动……我觉得改成“运动”这个词好一点，打死我都不愿意再去当面试官，大家都不愿意去。后来老板亲自出面跟我们说这个活动/运动的重要性，还说这是公司的战略级动作，要把目前60分的人慢慢换成70分的，然后从70分慢慢换成80分，进而再换成90分……开完会后，一个在公司做了四年，起初1K2月薪入职，至今未超过4K 的老员工没回到座位上，直接从会议室走去 HR 办公室领了离职单……

2、某年 CJ 前夕，公司打算去那儿摆个展台

我还算蛮配合工作的，把以前我在 CJ 上排到的各类照片拿出来给公司当素材，好方便联系展台设计公司来做这个单。

后来某总打算说我们也要出几个 SG，找了几个公司内还算水灵的妹子，起初她们答应了。后来不知道为什么，情况有些变化，我当时还天真地以为可能她们看到我照片中现场的场面太大，被吓到了所以不敢去。

我还跟他们解释其实不是你们所想的这样反正这些妹子都变成集体抗拒去 CJ，还有宣称要离职的。过了很久之后我才从其中一个人那里知道缘由……

3、内服其实就是高级客服，有的公司叫特服，专门为一些大额 RMB 客户服务的。一般男内服就会被分配到女玩家，女内服就会分配男玩家，这个不绝对，做这一岗位的都不太稳定，但是如果人手足够的话，通常都是这么安排的。

在后台其实可以看到对方的支付信息，银行卡之类的，有一些没有绑定身份证和手机号的，内服会主动在游戏里面联系他，经常会送一些小东西，什么 VIP 服务之类的鬼话，然后让他透露更多个人信息。

很多时候内服甚至会打电话跟大额们聊天，了解他们的情况，意图其实都是为了先把鱼养起来，等到合适的时候再去捞，一般这些账号如果多天不登录，排行滑落，内服甚至都会主动打电话过去询问情况，如果你手误强化把装备搞爆了，内服一般都有权限把东西还给你，总之就是各种普通人享受不到的服务

很多婚恋网站都有内部人员主动去贴客户的案例，其实游戏圈子也有

记得一个水灵的新应届生妹子，来到公司做客服，后来因为声线比较好听，被调去内服部，还没半年就辞职了，大伙儿都知道原因，后来过了很久，听客服同事八卦说，还记得那个谁谁谁么?她跑去某地找某某某被人玩几个月后弃置不理，现在又回来我们这个城市了，完全变了一个人什么什么的……

应届生总是这么的天真

原文链接：http://youxiputao.com/articles/783

[前端开发]

跨域方法汇总

做 Web 开发经常需要面对跨域问题，跨域问题的根源是浏览器安全中的同源策略，比如说，对于 http://www.a.com/1.html 来说：

http://www.a.com/2.html 是同源的；

https://www.a.com/2.html 是不同源的，原因是协议不同；

http://www.a.com:8080/2.html 是不同源的，原因是端口不同；

http://sub.a.com/2.html 是不同源的，原因是主机不同。

在浏览器中，<script>、、<iframe>和<link>这几个标签是可以加载跨域（非同源）的资源的，并且加载的方式其实相当于一次普通的 GET 请求，唯一不同的是，为了安全起见，浏览器不允许这种方式下对加载到的资源的读写操作，而只能使用标签本身应当具备的能力（比如脚本执行、样式应用等等）。

最常见的跨域问题是 Ajax 跨域访问的问题，默认情况下，跨域的 URL 是无法通过 Ajax 访问的。这里我记录我所了解到的跨域的方法：

1. 服务器端代理，这没有什么可说的，缺点在于，默认情况下接收 Ajax 请求的服务端是无法获取到的客户端的 IP 和 UA 的。

2. iframe，使用 iframe 其实相当于开了一个新的网页，具体跨域的方法大致是，域 A 打开的母页面嵌套一个指向域 B 的 iframe，然后提交数据，完成之后，B 的服务端可以：

返回一个 302 重定向响应，把结果重新指回 A 域；

在此 iframe 内部再嵌套一个指向 A 域的 iframe。

这两者都最终实现了跨域的调用，这个方法功能上要比下面介绍到的 JSONP 更强，因为跨域完毕之后 DOM 操作和互相之间的 JavaScript 调用都是没有问题的，但是也有一些限制，比如结果要以 URL 参数传递，这就意味着在结果数据量很大的时候需要分割传递，甚是麻烦；还有一个麻烦是 iframe 本身带来的，母页面和 iframe 本身的交互本身就有安全性限制。

3. 利用 script 标签跨域，这个办法也很常见，script 标签是可以加载异域的 JavaScript 并执行的，通过预先设定好的 callback 函数来实现和母页面的交互。它

有一个大名，叫做 JSONP 跨域，JSONP 是 JSON with Padding 的略称。它是一个非官方的协议，明明是加载 script，为啥和 JSON 扯上关系呢？原来就是这个 callback 函数，对它的使用有一个典型的方式，就是通过 JSON 来传参，即将 JSON 数据填充进回调函数，这就是 JSONP 的 JSON+Padding 的含义。

在互联网上有很多 JSONP 的服务来提供数据，本质上就是跨域请求，并且在请求 URL 中指定好 callback，比如 callback=result，那么在获取到这些数据以后，就会自动调用 result 函数，并且把这些数据以 JSON 的形式传进去，例如（搜索“football”）：

http://ajax.googleapis.com/ajax/services/search/web?v=1.0&q=football&callback=result

```
$.getJSON("http://ajax.googleapis.com/ajax/services/search/web?v=1.0&
1q=football&callback=?",function(data){
2        //...
3    });
```

使用 JQuery 来调用就写成：

总的来说，JSONP 的跨域方式的局限性在于，只能使用 GET 请求，并且不能解决不同域的两个页面之间如何进行 JavaScript 调用的问题。

4. Flash 跨域：

它会访问目标网站根目录下面的 crossdomain.xml 文件，根据文件中的内容来确定是否允许此次跨域访问：

```
1    <cross-domain-policy>
2        <allow-access-fromdomain="xxx.xxx.com"/>
3    </cross-domain-policy>
```

5. img 标签也可以使用，这也是一种非常常见的方法，功能上面弱一点，只能发送一个 get 请求，没有什么回调，Google 的点击计数就是这样确定的。

6. window.PostMessage，这个算是 HTML5 新加入的为跨域通讯考虑的机制，只有 Firefox 3、Safari 4 和 IE8 及之后的版本支持。使用它向其它窗口发送消息的

调用方式如下：

```
1       otherWindow.postMessage(message, targetOrigin);
```

在接收的窗口，需要设置一个事件处理函数来接收发过来的消息：

```
1       window.addEventListener("message", receiveMessage,false);
2       functionreceiveMessage(event){
3               if(event.origin !=="http://example.org:8080")
4                       return;
5       }
```

注意这里必需要使用消息的 origin 和 source 属性来验证发送者的身份，否则会造成 XSS 漏洞。

7. Access Control

有一些浏览器支持 Access-Control-Allow-Origin 这样的响应头，比如：

```
1       header("Access-Control-Allow-Origin: http://www.a.com");
```

就指定了允许对 www.a.com 跨域访问。

8. window.name

这个东西其实以前被用作黑客 XSS 的手段，其本质是，当 window 的 location 变化的时候，页面会重新加载，但是有趣的是，这个 window.name 居然不发生变化，那么就可以用它来传值了。配合 iframe，改变几次 iframe 的 window 对象，就完成了实用的跨域数据传递。

9. document.domain

这个方式适用于 a.example.com 和 b.example.com 这种跨域的通信，因为二者有一个共有的域，叫做 example.com，只要设置 document.domain 为 example.com 就可以了，但是如果 a.example1.com 和 b.example2.com 之间要通信，它就没办法了。

10. Fragment Identitier Messaging（FIM）

这个方法很有意思，也需要 iframe 的配合。Fragment Identitier 就是 URL 的井号（#）后面的经常用于锚点定位的部分，这部分的改变不会导致页面刷新，

母窗口可以随便访问 iframe 的 URL，而 iframe 也可以随便访问母窗口的 URL，那这二者之间就可以通过改变 Fragmement Identitier 来实现通信了。缺点是 Fragmement Identitier 的改变会产生不必要的历史记录，而且也有长度限制；另外，有的浏览器不支持 onhashchange 事件。

11. Cross Frame（CF）

这种方法是上述 FIM 方法的变种，CF 和 FIM 的本质其实在我的《GWT 初体验》这篇文章里面都有介绍（只不过是被用来实现历史和后退功能了），它会动态创建一个不可见的 iframe，指向异域，处理完以后，这个 iframe 的 URL 中的 Fragment Identitier 包含了处理结果，供母页面访问，而浏览器的 URL 没有任何变化。

12. Cookie+P3P 协议

利用 P3P 协议下跨域访问 Cookie 的特性，来实现跨域访问，也算一奇招。P3P 是 W3C 公布的一项隐私保护推荐标准，旨在为网上冲浪的 Internet 用户提供隐私保护。把 Cookie 的 path 设置为“/”，即没有任何域的限制，这个时候有的浏览器下面允许别的 URL 的页面来读取，有的则不允许，这种情况下需要在母页面响应的头上面设置 P3P 的头：

P3P: CP="CURa ADMa DEVa PSAo PSDo OUR BUS UNI PUR INT DEM STA 1PRE COM NAV OTC NOI DSP COR"

原文链接：http://www.raychase.net/2216

JavaScript 多文件下载

对于文件的下载，可以说是一个十分常见的话题，前端的很多项目中都会有这样的需求，比如 highChart 统计图的导出，在线图片编辑中的图片保存，在线代码编辑的代码导出等等。而很多时候，我们只给了一个链接，用户需要右键点击链接，然后选择“另存为”，这个过程虽说不麻烦，但还是需要两步操作，倘若用户想保存页面中的多个链接文件，就得重复操作很多次，最常见的就是英语听力网站上的音频下载，手都要点麻！

本文的目的是介绍如何利用 javascript 进行多文件的下载，也就是当用户点击某个链接或者按钮的时候，同时下载多个文件。这里的“同时”用的不是很准确，在现代浏览器中可以实现多文件的并行下载，而在一些老版本浏览器，如 IE8-，此类的浏览器就进行只能单个文件的下载，但是我们可以让多个文件依次保存下来，算是串行下载吧~

一、文件类型介绍及其特点

1. 一般类型

平时比较常见的有 txt、png、jpg、zip、tar 等各种文件格式，这些文件格式中，一部分浏览器是会直接打开链接显示内容的，而另外一部分，浏览器不识别响应头，或者不能解析对应的格式，于是当做文件直接下载下来了。如：

file

这句代码，若直接点开链接，浏览器将会直接下载该文件。

2. dataURL 类型

dataURL 也是十分常见的类型，他可以作为 src 或者 url() 的参数送进去。比较常见的有如下几种：

文本： data:text/plain;这里是正文内容。

图片：

....

base64 是用的比较广泛的一种数据格式。

Base64 格式

data:[][;charset=][;base64],

Base64 在 CSS 中的使用：

.demoImg{background-image:
url("...."); }

Base64 在 HTML 中的使用：

<imgwidth="40"height="30"
src="...." />

3. Blob 流

Blob 对象表示不可变的、包含原始数据的类文件对象。具体的内容可以参阅 MDN 文档。

他的使用也是特别的方便，如：

```
var aFileParts = ['<a id="a"><b id="b">hey!</b></a>'];
var oMyBlob = new Blob(aFileParts, {type : 'text/html'}); // the blob
```

Blob 接收两个参数，一个是数组类型的数据对象，他可以是 ArrayBuffer、ArrayBufferView、Blob、String 等诸多类型；第二个参数是 MINE 类型设置。而本文我们要用到的是 URLcreateObjectURL() 这个函数，他的作用是将一个 URL 所代表的内容转化成一个 DOMString，产生的结果是一个 文件对象 或者 Blob 对象。

4. 二进制流

我们利用 File API 读取文件的时候，拿到的是数据的二进制流格式，这些类型可以直接被 ArrayBuffer 等接收，本文中没有用到，就不细说了。

二、JavaScript 多文件下载

HTML5 中 a 标签多了一个属性——download，用户点击链接浏览器会打开并显示该链接的内容，若在链接中加了 download 属性，点击该链接不会打开这个文件，而是直接下载。虽说是比较好用，但低版本浏览器不兼容，这个在本届的 2 和 3 中将会讲到解决方案。

在这里，我们可以利用属性检测来判断浏览器类型：

```
h5Down = document.createElement("a").hasOwnProperty("download");
```

1. a 标签 download 属性的使用

利用 download 属性可以直接下载单个文件，若想点击一次下载多个文件，

就得稍加处理下了：

```
function downloadFile(fileName, content){
var aLink = document.createElement("a"),
evt = document.createEvent("HTMLEvents");
evt.initEvent("click");
aLink.download = fileName;
aLink.href = content;
aLink.dispatchEvent(evt);
}
```

download 属性的作用除了让浏览器忽略文件的 MIME 类型之外，还会把该属性的值作为文件名。你可以在 chrome 控制台运行这句程序：

```
downloadFile("barretlee.html", "./");
```

浏览器会提示是否保留（下载）该 html 文件。之前我们提到文件类型还可能是 dataURL 或者是 Blob 流，为了让程序也支持这些数据类型，稍微修改下上面的函数：

```
function downloadFile(fileName, content){
var aLink = document.createElement('a');
, blob = new Blob([content])
, evt = document.createEvent("HTMLEvents");
evt.initEvent("click");
aLink.download = fileName;
aLink.href = URL.createObjectURL(blob);
aLink.dispatchEvent(evt);
}
```

new Blob([content])，现将文件转换成一个 Blog 流，然后，使用 URL.createObjectURL()将其转换成一个 DOMString。这样我们就支持 data64 和其他数据类型的 content 了~

2. window.open 之后 execCommand("SaveAs")

上面也提到了，尽管 download 属性是十分便利的 H5 利器，但低版本 IE

根本不赏脸，要说方法，IE 还是有很多方式去转换的，比如 ADOBE.STREAM 的 activeX 对象可以把文件转换成文件流，然后写入到一个要保存的文件中。这里要谈到的是略微方便一点的方式：先把内容写到一个新开的 window 对象中，然后利用 execCommand 执行保存命令，就相当于我们在页面上按下 Ctrl+S，这样页面内的信息都会 down 下来。

```
// 将文件在一个 window 窗口中打开，并隐藏这个窗口。
var win = window.open("path/to/file.ext", "new Window", "width=0,height=0");
// 在 win 窗口中按下 ctrl+s 保存窗口内容
win.document.execCommand("SaveAs", true, "filename.ext");
// 使用完了，关闭窗口
win.close();
```

这个过程十分明了，不过这里会存在一个问题，并不是程序的问题，而是浏览器的问题，如果我们用 搜狗浏览器 或者 360 浏览器 打开新窗口的话，他会新开一个标签页，而不是新开一个窗口，更可恶的时候部分浏览器不允许 window.open （这个可以设置）。所以只好另觅他法了。

3. iframe 中操作

既然新开一个窗口那么麻烦，我就在本窗口下完成工作~

```
function IEdownloadFile(fileName, contentOrPath){
var ifr = document.createElement('iframe');
ifr.style.display = 'none';
ifr.src = contentOrPath;
document.body.appendChild(ifr);
// 保存页面 -> 保存文件
ifr.contentWindow.document.execCommand('SaveAs', false, fileName);
document.body.removeChild(ifr);
}
```

一般的链接我们可以直接给 iframe 添加 src 属性，然后执行 saveAs 命令，倘若我们使用的是 data64 编码的文件，这个怎么办？

```
var isImg = contentOrPath.slice(0, 10) === "data:image";
```

// dataURL 的情况

isImg && ifr.contentWindow.document.write("<img src='" +

contentOrPath + "' />");

这个也比较好处理，直接把文件写入到 iframe 中，然后在执行保存。

三、代码的封装与接口介绍

1. 代码的封装以及相关 DEMO

封装：lib.js DEMO：javascript-multiple-download

2. 接口的调用

提供了三个接口，支持单文件下载，多文件下载，多文件下载自定义命名。

1）单文件下载

Downer("./file/test.txt");

2）多文件下载

Downer(["./file/test.txt","./file/test.txt"]);

3）多文件下载自定义命名

Downer({

"1.txt":"./file/test.txt",

"2.jpg":"./file/test.jpg"

});

文件的 URL 如 ./file/test.txt 都可以改成 base64 或者其他格式。

四、服务器支持与后端实现

1. 后端实现

后端实现的原理，就是在响应头中加入一些特殊的标记，如前端发送这样的请求：

function download(path) {

var ifrm = document.getElementById(frame);

ifrm.src = "download.php?path="+path;

}

后端的响应为

<?php

```
header("Content-Type: application/octet-stream");
header("Content-Disposition: attachment; filename=".$_GET['path']);
readfile($_GET['path']);
?>
```

告诉浏览器这是一个流文件，作为附件方式发送给你，请忽略 MINE type，直接保存。

2. 服务器配置

若后台是 apche 作为服务器，可以配置 htaccess 文件：

```
<filesmatch "\.(zip|rar)$"="">
Header set Content-Disposition attachment
</filesmatch>
```

意思是只要请求的是 zip 或者 rar 类型的文件，那么久添加一个 Content-Disposition:attachment 的响应头。这样就可以在 php 代码中省略麻烦的操作。

四、小结

由于行文仓促，文中会有不少错误，对多文件下载有更好的提议，希望提出来共同分享！

原文链接：http://barretlee.com/multiple-download-with-javascript/

写一个更好的 Javascript DOM 库

目前，jQuery 是事实上的操作文档对象模型（DOM）的库。它可以与流行的客户端 MV*框架结合使用，并且拥有大量的插件与大型的社区。开发者对于 Javascript 的兴趣与日俱增的同时，很多人开始好奇，原生的 API 是如何工作的，以及我们何时应该直接使用它们而不是引用一个额外的库。

最近，我开始发现越来越多的 jQuery 的问题，至少是在我的使用中是这样的。其中的绝大多数涉及到 jQuery 的核心，在不取消向后兼容的情况下无法解决——而向后兼容又非常重要。与很多人一样，我继续使用了它一段时间，每天浏览所有讨厌的浏览器怪异模式。

后来, Daniel Buchner 创造了 SelectorListener，于是有了“live 扩展（live extensions）”的概念。我开始考虑创造一系列的函数，使得我们可以使用比迄今为止用过的方法都更好的方式来创建非干扰性的 DOM 组件。目标是回顾已有的 API 与解决方案，并创造一个更干净、可测试且轻量级的库。

向库添加有用的特性

是 live 扩展的想法鼓励我开发了 better-dom 项目，不过，还有一些其他的有趣的特性使得它成为一个独特的库。我们快速地看一下：

live 扩展

原生的动画

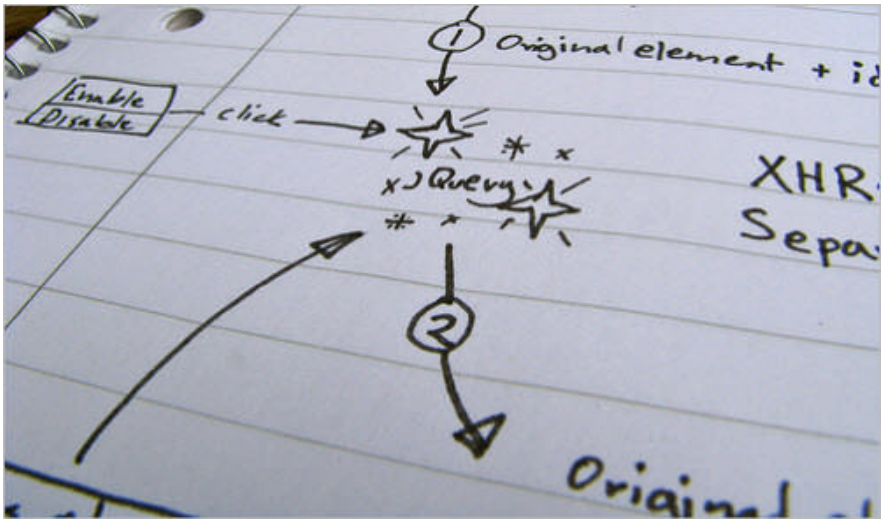
内置的微模板

国际化的支持

live 扩展

jQuery 有一个叫做“live 事件（live events）”的概念。借助事件代理，它使得开发者可以处理现有的以及未来的元素。但是许多情况会要求更大的灵活度。比如为了初始化一个部件而需要对 DOM 进行转换，事件代理就会力不从心。故而，live 扩展。

目标是，只需定义一次扩展并使得所有未来的元素快速略过初始化函数，而无论部件的复杂度。这个很重要，因为它使得我们可以声明式地开发 web 页面，从而在 AJAX 应用中表现优异。



Live 扩展使得你无需调用初始化方法就可以操作未来的元素

我们来看一个简单的例子。假设我们的任务是实现一个完全自定义的提示框。:hover 伪类选择器并无帮助，因为提示框的位置随着鼠标移动而变化。事件代理也不是很合适；监听文档树中所有元素的 mouseover 及 mouseleave 事件代价很大。live 扩展将拯救你！

```
1       DOM.extend("[title]", {
2           constructor:function() {
3               vartooltip = DOM.create("span.custom-title");
4
5               // set the title's textContent and hide it initially
6               tooltip.set("textContent",this.get("title")).hide();
7
8               this
9                   // remove legacy title
10                  .set("title",null)
11                  // store reference for quicker access
12                  .data("tooltip", tooltip)
13                  // register event handlers
14                  .on("mouseenter",this.onMouseEnter, ["clientX","clientY"])
15                  .on("mouseleave",this.onMouseLeave)
```

```
16                  // insert the title element into DOM
17                  .append(tooltip);
18          },
19          onMouseEnter:function(x, y) {
20              this.data("tooltip").style({left: x, top: y}).show();
21          },
22          onMouseLeave:function() {
23              this.data("tooltip").hide();
24          }
25      })
```

我们可以在 CSS 中定义 .custom-title 元素的样式：

```
1       .custom-title {
2           position:fixed;/* required */
3           border:1pxsolid#faebcc;
4           background:#faf8f0;
5       }
```

当你向页面中插入一个带 title 属性的新元素时，最有趣的部分发生了。自定义的提示框无需调用任何初始化方法即可生效。

live 扩展是独立的；这样它们并不需要为了使得未来的内容生效去调用一个初始化方法。因此它们可以与任何 DOM 库结合使用，将 UI 代码分割成许多小的独立的块，从而简化应用的逻辑。

最后，同样很重要的，一些关于 Web 组件的内容。规范的一部分，“装饰器”，意在解决类似的问题。目前，它使用了一种基于标记的实现，通过特殊的语法，将事件监听者绑定到子元素上。但它仍只是早期的草案：

“装饰器，与 Web 组件的其它部分不同的是，它还没有一个规范。”

原生动画

多亏了 Apple, CSS 现在拥有了对动画的良好支持。过去动画通常使用 Javascript 的 setInterval 及 setTimeout 实现。这曾经是很酷的特性——但是现在看来，它更像是坏的实践。原生的动画总是更平滑：常常更快，开销更小，并且在浏览器不支持的情况下可以很好地降级。

better-dom 中，没有 animate 方法：只有 show, hide 以及 toggle。库使用基于标准的 aria-hidden 属性来在 CSS 中获取一个隐藏元素的状态。

为了说明它是如何工作的，我们来为先前介绍的提示框添加一个简单的动画效果：

```
1       .custom-title {
2           position:fixed;/* required */
3           border:1pxsolid#faebcc;
4           background:#faf8f0;
5           /* animation code */
6           opacity:1;
7           -webkit-transition: opacity0.5s;
8           transition: opacity0.5s;
9       }
10
11      .custom-title[aria-hidden=true] {
12          opacity:0;
13      }
```

show() 以及 hide() 在内部将 aria-hidden 属性值设置为 false 或 true。这使得 CSS 可以处理动画与转换。

你可以在这个 demo 中看到更多使用了 better-dom 的动画。

内置的微模板

HTML 字符串冗长而繁琐。寻找替代的过程中我发现了超棒的 Emmet。如今 Emmet 已经是一个非常流行的文本编辑器插件，它拥有漂亮而简洁的语法。比

如这段 HTML：

```
1 body.append("<ul><li class='list-item'></li><li class='list-item'></li><li
  class='list-item'></li></ul>");
```

与对应的微模板比较：

```
1 body.append("ul>li.list-item*3");
```

在 better-dom 中，任何接受 HTML 的方法同样接受 Emmet 表达式。缩写解析器很快，所以不用担心付出性能代价。如果需要，还有一个模板预编译函数可用。

国际化支持

开发一个 UI 组件经常会需要本地化——这并不轻松。多年来，很多人使用不同的方法解决这个问题。在 better-dom 中，我相信改变 CSS 选择器的状态，就如同转换语言。

从概念上说，转换语言正是改变内容的“表现”。在 CSS2 中，有几个伪类选择器可用于描述这样一个模型：:lang 以及:before。我们来看下边的代码：

```
1 [data-i18n="hello"]:before {
2 content: "Hello Maksim!";
3 }
4
5 [data-i18n="hello"]:lang(ru):before {
6 content: "Привет Максим!";
7 }
```

这是个很简单的把戏：html 元素的 lang 属性控制当前语言，而 content 属性值根据当前的语言变化。通过使用如 data-i18n 这样的属性，我们可以在 HTML 中维护文本内容。

```
1 [data-i18n]:before {
2 content: attr(data-i18n);
3 }
4
5 [data-i18n="Hello Maksim!"]:lang(ru):before {
```

```
6           content: "Привет Максим!";
7       }
```

当然，这样的 CSS 并不吸引人，所以 better-com 提供了两个帮助方法：i18n 及 DOM.importStrings。前者用于更新 data-i18n 属性为合适的值，后者为特定的语言本地化字符串。

```
1       label.i18n("Hello Maksim!");
2       // the label displays "Hello Maksim!"
3       DOM.importStrings("ru",   "Hello Maksim!", "Привет Максим!");
4       // now if the page is set to ru language,
5       // the label will display "Привет Максим!"
6       label.set("lang", "ru");
7       // now the label will display "Привет Максим!"
8       // despite the web page's language
```

还可以使用参数化的字符串。直接向关键字符串中添加${param} 变量：

```
1       label.i18n("Hello ${user}!", {user:"Maksim"});
2       // the label will display "Hello Maksim!"
```

让原生的 APIs 更加优雅

通常我们都希望遵从标准。但是有时候标准对用户并不友好。DOM 就是一团糟 ，为了将其变得好用，我们不得不把它包装到一个方便的 API 中。尽管开源的库已经作了很多改进，仍有一些部分可以做得更好：

getter 及 setter

事件处理

功能性的方法支持

GETTER 及 SETTER

原生的 DOM 元素有 attributes 及 properties 的概念，但他们的行为并不完全一致。假设我们在一个 web 页面中有如下的标记：

```
<ahref="/chemerisuk/better-dom"id="foo"data-test="test">better-dom</
a>
```

为了解释为什么“DOM 就是一团糟”，我们来看这：

```
varlink = document.getElementById("foo");

link.href;// => "https://github.com/chemerisuk/better-dom"
link.getAttribute("href");// => "/chemerisuk/better-dom"
link["data-test"];// => undefined
link.getAttribute("data-test");// => "test"

link.href ="abc";
link.href;// => "https://github.com/abc"
link.getAttribute("href");// => "abc"
```

一个 attribute 与其在 HTML 中对应的字符串相等，但元素的同名 property 可能会有一些奇怪的行为，比如在上边列出来的，生成完全合格的 URL。这些区别有时会导致混淆。

在实际使用中，很难想像一个这样的区别有用的场景。除此之外，开发者必须总是牢记哪些值（attribute 或 property）被使用了，这会引入没必要的复杂度。

在 better-dom 中，事情要清楚得多。每个元素都只有智能的 getter 与 setter。

```
varlink = DOM.find("#foo");

link.get("href");// => "https://github.com/chemerisuk/better-dom"
link.set("href","abc");
link.get("href");// => "https://github.com/abc"
link.get("data-attr");// => "test"
```

首先，它做一次属性（property）查找，如果是已定义的，则返回供操作。不然，getter 及 setter 作用于对应的元素属性（attribute）。对于 boolean 值 (checked, selected, 这些), 可以直接使用 true 或 false 来更新值：改变元素的该属性（property）将触发对应的 attibute（原生行为）的更新。

改良的事件处理

事件处理是 DOM 中很重要的一部分，然而，我发现一个根本性的问题：将 event 对象传入元素监听者的行为导致关心可测试性的开发者不得不伪造第一个参数（事件对象），或是创建一个额外的函数来传入事件处理函数仅需的事件属性。

```
1    varbutton = document.getElementById("foo");
2
3    button.addEventListener("click",function(e) {
4        handleButtonClick(e.button);
5    },false);
```

这很烦人。不过如果我们将可变部分抽象为一个参数，我们就可以摆脱额外的函数：

```
1    var button = DOM.find("#foo");
2
3    button.on("click", handleButtonClick, ["button"]);
```

默认地，事件处理函数会被传入["target", "defaultPrevented"] 数组，所以不用为了获得这些属性添加最后一个参数。

```
1    button.on("click",function(target, canceled) {
2        // handle button click here
3    });
```

延时绑定也得到了支持(我建议读一下 Peter Michaux 关于这个主题的回顾)。它是 W3C 的标准中常规事件绑定的更加灵活的替换物。它在你需要频繁进行启用和关闭方法调用时非常有用。

```
1    button._handleButtonClick =function() { alert("click!"); };
2
3    button.on("click","_handleButtonClick");
4    button.fire("click");// shows "clicked" message
5    button._handleButtonClick =null;
6    button.fire("click");// shows nothing
```

最后，同样很重要的，better-dom 不提供任何对于遗留的或不同浏览器中表现不一致的 API 的调用，比如 click(), focus() 和 submit()。 调用他们的唯一方式是使用 fire 方法，它在没有监听者返回 false 的情况下执行默认行为：

```
1    link.fire("click");// clicks on the link
2    link.on("click",function() {returnfalse; });
3    link.fire("click");// triggers the handler above but doesn't do a click
```

功能性方法的支持

ES5 规范了一些的有用的数组方法，包括 map, filter 以及 some。它们允许我们以符合标准的方式使用通用的集合操作。因此现在我们有了诸如 Underscore 和 Lo-Dash 这样的项目，它们在老的浏览器上实现这些方法。

better-dom 中的每个元素（或集合）都内置了如下的方法：

each (它与 forEach 的区别在于返回 this 而不是 undefined)

some

every

map

filter

reduce[Right]

```
1    varurls, activeLi, linkText;
2
3    urls = menu.findAll("a").map(function(el) {
4        returnel.get("href");
5    });
6    activeLi = menu.children().filter(function(el) {
7        returnel.hasClass("active");
8    });
9    linkText = menu.children().reduce(function(memo, el) {
10       returnmemo || el.hasClass("active") && el.find("a").get()
11   },false);
```

避免 jQuery 的问题

在不放弃向后兼容的情况下，以下的绝大多数问题无法在 jQuery 中得到解决。这是为什么创造一个新的库看起来是合乎逻辑的解决途径。

“神奇的” $ 函数

[] 操作符的值

关于 return false 的问题

find 以及 findAll

“神奇的” $ 函数

每个人都或多或少听说过$ (美元) 函数的神奇。一个单字符的名字并不具有描述性，所以它看起来像是一个内置的语言操作符。这也正是缺乏经验的开发者的代码中$的调用随处可见的原因。

在背后的实现中，$是个极其复杂的函数。经常地执行，尤其是 mousemove 、scroll 这样的频繁事件中，会导致较差的 UI 性能。

尽管有很多文章建议将 jQuery 对象缓存下来，开发者依旧在将$大量嵌入在代码中，因为 jQuery 库的语法鼓励了这样的代码风格。

$函数的另一个问题是，它可以被用来做完全不同的两件事。人们已经喜欢了这样的语法，但通常来说，这是一个失败的函数设计：

```
1    $("a");// => searches all elements that match “a” selector
2    $("<a>");// => creates a <a> element with jQuery wrapper
```

better-dom 使用了几个函数来承担 jQuery 中$函数的职责：find[All] 以及 DOM.create。find[All] 被用来依据 CSS 选择器来获取元素。 DOM.create 在内存中创建一个新的节点树。它们的名字就可以清晰地表明它们的职责。

[]操作符的值

导致$函数被频繁调用的另一个原因正是方括号操作符。当一个新的 jQuery 对象被创建的时候，所有相关的节点都被存储在数值型属性中。但是请注意，这样一个数值属性的值包含了一个原生的元素实例（而非经 jQuery 包装过的对象）：

```
1    varlinks = $("a");
2
3    links[0].on("click",function() { ... });// throws an error
4    $(links[0]).on("click",function() { ... });// works fine
```

正因为这样的特性，jQuery 或是其它库（比如 Underscore）中的每一个功能方法都要求当前元素在回调函数中使用$() 包起来。因此，开发者必须时刻牢记他们正在操作的对象类型——一个原生元素或是一个包装过的对象——尽管事实上他们正在使用一个操作 DOM 的库。

在 better-dom 中，方括号操作符返回一个库对象，所以开发者可以忘记原生元素。只有一种可接受的方式来获取原生元素：使用一个特别的 legacy 方法。

```
1    varfoo = DOM.find("#foo");
2
3    foo.legacy(function(node) {
4      // use Hammer library to bind a swipe listener
5      Hammer(node).on("swipe",function(e) {
6        // handle swipe gesture here
7      });
8    });
```

事实上，只有非常少见的情况会需要这个方法，比如兼容一个原生的方法，或是另一个 DOM 库（比如上边例子中的 Hammer）。

关于 RETURN FALSE 的问题

jQuery 事件处理函数中返回 false 后的奇怪的拦截行为让我一直很纠结。依据 W3C 的标准，它应该在大多数情况下取消默认行为。在 jQuery 中，return false 还会阻止事件代理。

这样的捕获会导致问题：

1 自行调用 stopPropagation() 可能导致兼容性问题，因为它阻止了其他任务相关的监听者的执行。

2 大部分开发者（即使是一些有经验的）并没有意识到这样的行为

尚不清楚为什么 jQuery 社区决定不遵循标准。但 better-dom 并不会重蹈覆辙。 所以，正如每个人所预期的，事件句柄中的 return false 只会阻止浏览器默认行为，而不会干扰事件冒泡。

FIND 以及 FINDALL

元素查找是在浏览器中代价最大的操作之一。两个原生的方法可以用来实现

它：querySelector 以及 querySelectorAll。区别在于前者在匹配到第一个结果后即停止查找。

这个特性使得我们可以显著减少特定情形下的迭代次数。在我的测试中，速度提升到了二十倍!而且，可以预见，依据文档树的规模，提升可能达到更多。

jQuery 提供了一个 find 方法，使用 querySelectorAll ，用于一般的情形。目前还没有函数使用 querySelector 来只获取第一个匹配的元素。

better-dom 库有两个单独的方法：find 及 findAll。它们允许我们使用 querySelector 优化。为了评估潜在的性能提升，我在我上一个商业项目的所有源代码中搜索了这些方法的使用：

find

在 11 个文件中匹配 103 次

findAll

在 4 个文件中匹配 14 次

很明显 find 方法要受欢迎得多。这说明 querySelector 优化在大多数情况下是有意义的，并能推动相当的性能提升。

结论

live 扩展确实使得解决前端问题简单不少。将 UI 分割为许多小块可以带来更加独立、可维护的解决方案。不过正如我们所展示的，一个框架不仅仅是关于这些（尽管这是主要目标）。

我在开发过程中学习到的一件事是，如果你不喜欢某个标准，或者你对该如何做某件事情有自己不同看法，那么就去实现它，证明你的方法可行。这也很有趣！

原文链接：http://blog.jobbole.com/58169/

Hybrid APP 开发者一定不要错过的框架和工具

ionicFramework

我是 hybrid app 的忠实粉丝和大力倡导者，从新浪移动云开始就不断的寻找能帮助 Web 程序员开发出漂亮又好用的 UI 层框架。在历经了 jqmobile、sencha touch 等框架后，一直没能找到一个真正符合我的想法的框架：它应该为 hybrid app 设计、组件化结构、UI 简洁而优美。

很多同学不明白为 Hybrid app 设计跟为 Mobile web 设计有什么不同，我只说最典型的一点：Hybrid App 的资源都在本地的，没有网络读取的消耗，所以最理想的方式是一次性载入多个界面，应用直接在多个界面之间切换，而不是为了节省流量通过 ajax 按需载入。后者正是 jqm 等框架的做法。sencha touch 在概念层上我接受不了，它让我感觉自己不是在写界面，而是在做算法作业。关于这些老框架的吐槽，以前写过，就不再多说。接下来说说最近的新发现。

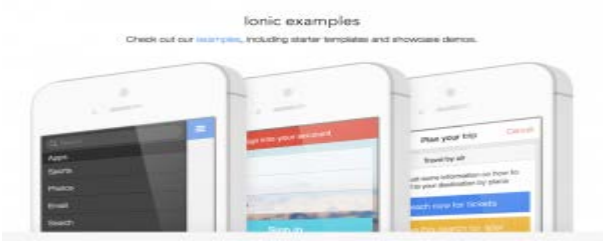
转眼快两年，很多更适合 hybrid app 的框架开始出现。这次选型我其实先选了 chocolatechip-ui 。



它和我之前写过的 lazymobile 理念很类似，采用 div 来作为 App 界面，界面之间的切换其实就是浏览器在 div 上的滑动。界面采用了 ios7 的平面设计风格，很讨喜。我看到就心动了，读完基本代码实现后就找了个小需求来试用。那是一个 tab+side menu 的 app，然后我发现 chocolate-chip 的组件的封装程度不够，多个组件之间重复嵌套时会出现各种互相影响。这显然违背了正交性原则。于是我依依不舍的和它分了手。

然后我遇到了 ionicframework。ionic 采用 angularjs 作为其基础，这就在封装性上有了质的提升。你可以使用类似<weibo>的标签来描述一条微博。它也同样采用 angualar-ui 里边的 states 来切换界面，从而保证了切换过程的平滑。另外就是它还直接整合了 Cordova(就是 phonegap 了)的命令行工具，写完后直接一个命

令就可以编译 app 了。



ionic 的学习成本比较高。因为 Angularjs 是一个真正的 MVC 框架，它的 M 和 V 双向绑定。我春节花了点时间学了一下，能写一些简单应用了。学习的时候有时候还是思维转换不过来，比如我曾苦苦思索 form 表单要怎么提交，提交到哪里去。但实际上 form 表单的控件本来就是绑定到数据对象的，只要调用数据对象的 save 方法就好了。

但整体来讲我还是很推荐这个框架的，而且我也认为 AngularJS 的做法代表了未来，学习一些新东西有时候会让人生更多乐趣。

Ripple Emulator

在编写代码的时候，我一般用 node 的 http-server 直接在代码目录起一个 web 服务，用浏览器调试。但是代码里边那些 phonegap 的扩展功能往往会导致页面报错。之前为了解决这个问题，我们做了云窗调试器，但现在因为某些原因，基本不更新了。另外云窗调试器需要将代码传到 SAE 上后才能调试，也没有本地来得便捷。

这里要推荐给大家的是一个 Chrome 扩展，Ripple Emulator。

用了这个扩展，你就可以直接在 Chrome 上调试 Colrdova 的功能了。能选择模拟器来测试屏幕大小，还能选择模拟器的 Cordova 版本。五星推荐，用了你会回来点赞的。

还有一些 Angular 相关的工具我也在用，这次就先不整理了。以后有空再慢慢整理吧，请关注我的微博获取最新的消息。

原文链接：http://ftqq.com/2014/02/the-missing-tools-for-hybrid-app-developer/

[编程语言]

Python 超级程序员使用的开发工具

我以个人的身份采访了几个顶尖的 Python 程序员，问了他们以下 5 个简单的问题：

当前你的主要开发任务是什么？

你在项目中使用的电脑是怎样的？

你使用什么 IDE 开发？

你将来的计划是什么？

有什么给 Python 程序员的建议？

就是这几个问题，我找了几个顶尖的程序员和编程书籍作家，问他们这几个相同的问题。下面是他们的回答，希望在他们的回答中你能找到一些可以让你的开发更便捷的工具。

人物简介：Alex Martelli 是一位意大利计算机软件程序员，他是 Python 软件基金会的成员。从 2005 年初开始，他以“Über 技术首领”的身份在加利福尼亚景山谷歌总部工作。他拥有意大利 Bologna 大学电子工程学位(1980)。他是《Python in a Nutshell》一书的作者，他还和其他人合著了《Python Cookbook》，他还写了一些其它一些作品，大部分都是和 Python 相关的。因为他对 Python 社区杰出的贡献，他被授予的荣誉包括 2002 Activators’ Choice Award 和 2006 Frank Willison award。

你可以通过他的博客了解更多关于他的信息。

问题一、当前你的主要开发任务是什么？

给我的老板(谷歌)开发商业软件，大部分是商业智能方面。

问题二、你在项目中使用的硬件机器是怎样的？

基本上都是谷歌的云服务器，通过谷歌 APP 引擎直接或间接的使用。有时候也使用一个 Unix 工作站或 Macbook Air，但很少。

问题三、你使用什么 IDE 开发？

vim, gvim, mvim；偶尔用一下 iPython，但这好像不能算是 IDE 吧。

问题四、你将来的计划是什么？

应该还会待在谷歌——具体的研究方向那要看实际情况。

问题五、有什么给 Python 程序员的建议？

学习一下 App Engine(通过 webapp2 或 flask, jinja2, &c 以及其它类似 django 的轻量级环境)：它们大部分都会对你的 web 项目有很好的用处——如果你想在这些环境之外的地方部署 web 应用，Appscale 会是你最方便的选择。

Daniel Greenfeld



Daniel Greenfeld

人物简介： Daniel Greenfeld 是《Two Scoops of django》一书的合著人。他有超过 6 年的在 Django 领域的研究经验。之前他在美国宇航局做开发，就是在那里他开始了他的 Python 之旅，一直到现在。他目前是 Cartwheel Web 的负责人。

你可以通过他的 Twitter: @pydanny 和他的博客获取他的更多信息。

问题一、当前你的主要开发任务是什么？

我现在的工作主要是客户端方面，基本上都是用 Python。内容服务方面有时候会用到 Django，但当前的主要工作基本上跟 web 无关。

问题二、你在项目中使用的硬件机器是怎样的？

硬件方面，我使用一个 2011 年的 Macbook Air。因为我的开发方式，像 Macbook Air 这样的超级本用起来非常舒服。可苹果的产品一般都很贵，我就自己配了一个，我打算所有的任务都用它来做。如果哪天我能有一个新的笔记本，我想试试 Linux 的超级本或 Window8 的平板(以前我是靠虚拟机运行 Windows8)。

问题三、你使用什么 IDE 开发？

我使用 Sublime Text。我更喜欢文本编辑器，而不是 IDE，因为我喜欢深入研究程序库和它们的文档，而不是通过 IDE 工具来提示我。

问题四、你将来的计划是什么？

我未来的计划就是编程，锻炼，做我最擅长的事情。也许会去一个新地方旅游。我喜欢去没有去过的地方，认识新朋友，品尝没吃过的食物。但大部分我的旅游都是基于客户的需求，今年我不知道什么时候会有这样的机会。

问题五、有什么给 Python 程序员的建议？

我给 Python 程序员的建议是挑战极限。寻找新的代码库，研究它们，使用它们。它们有些会不是很好，不好的东西也会教会你一些东西(不该做的事情)。

Miguel Grinberg



Miguel Grinberg

人物简介： Miguel 是一个程序员，摄影师和电影制作人。生活在美国俄勒冈州。他目前正在写一本关于使用 Python 和 Flask 做 web 开发的书，将会由 O’Reilly Media 出版。这本书的官方网站是 Flask Book.

你可以通过他的 Twitter: @miguelgrinberg 和他的博客获取更多信息。

问题一、当前你的主要开发任务是什么？

也许你会吃惊，工作中我并不做 web 开发。我在为 Harmonic 做视频软件。C++是我在工作中使用的主要语言，但我们却使用了一个古老的用 Python 写的自动化单元测试框架，用起来很有趣。

空闲时间里我目前的第一大任务就是给 O'Reilly 出版社写一本关于 Flask 为框架的书，基本上快写成了。

问题二、你在项目中使用的硬件机器是怎样的？

我的机器是一个双系统的台式机，装的是 Ubuntu 和 Windows8.1。我还有一个 Mac OS X 笔记本。

我在跨平台方面有些变态。除了给 Arduino 这样的嵌入式系统写代码外，所有其它我写的代码(不管什么语言)，都需要在三种平台上运行。为了达到这个目的，我会不断的在 Linux，Windows 和 OS X 机器间切换。

问题三、你使用什么 IDE 开发？

我有一些最爱。PyCharm 非常棒，在大型项目时我越来越依赖它，主要是因为它的交互式的调试工具。

在 Windows 平台上我会使用 Visual Studio 的一个 Python 插件，免费的，非常出色，我估计大部分人没听说过它。

我还经常使用 SublimeText，通常是在笔记本上使用它。

Notepad++是我在 Window 台式机上最顺手的工具。在我所有的电脑上都装有一个非常老的 Vim，即使在 Windows 上也使用 Cygwin 运行它。

问题四、你将来的计划是什么？

我在四月份的 PyCon 大会上会有一个关于 Flask 的演讲，我下一步开始着手准备材料，因为书已经快写好了。

我最近没怎么写博客，我很想写一点。我想能多腾出一些时间写博客。我已经拟了一份话题目录，准备按着这个清单写。如果需要的话，我可以把里面的一个关于 Flask 上的 web sockets 的文章提到最前面。 

问题五、有什么给 Python 程序员的建议？

我对所有程序员——不光是 Python 爱好者——的建议是，多去实践，然后把它们分享给世界。分享很重要，从别人那获得反馈是让你提高的好方法。

而对于 Python 程序员我想说的是，如果我们只需要用嘴去讨论 Python2 和 Python3 的差别、而不需要花时间将众多的软件移植到 Python3，那该多好。玩笑。

说的太好了！

揭开其他程序员的面纱，窥视他们如何施展魔法工作，这很有用。你可以看到，他们使用的工具、技术，我们都可以采用，很多都是免费的。我很感谢他们能分享这些信息。

原文链接：http://www.aqee.net/tools-used-by-python-super-stars/

基于 Capistrano 工具的 Rails 程序部署方案

什么是 Capistrano？

简单地讲，Capistrano 是一个 gem。对于不太熟悉 Ruby 的朋友来说，gem 就是一套按照一定规范打包好的 Ruby 代码，可以类比 Java 里面的 Package。

实际上，Capistrano 是专门用来部署 rails 程序的一套工具。

为什么用它？

Rails 应用程序的部署方案十分灵活，适应于不同的生产服务器场景。在实际生产中，rails 程序中包含的各类 gem 和生产服务器上的 gem 往往版本不一致，这就为部署增加了一些难度。

利用 Capistrano 进行部署，就能很容易解决这个问题。同时，使用 Capistrano 部署 rails 程序，更有一种“敏捷”的感觉。

在进行 Rails 开发过程中，一个重要的工具就是版本控制，而我们经常用 Git。用过 Heroku 的开发者，或者有静态博客的朋友都能感受到一点——把版本控制与部署方案融合到一起非常方便。幸运的是，Capistrano 对包括 Git 在内的各类版本控制有非常好的支持。本文中，我将使用 Git 作为版本控制软件，与 Capistrano 一起工作，进行项目部署。

如何部署？

利用 Capistrano 部署 Rails 程序，可以分为下面四个步骤。

配置部署服务器

对应用程序进行版本控制

远程部署应用程序

版本控制推进部署

配置部署服务器

对生产服务器的基本配置，包括 Web 静态服务器、数据库、版本控制搭建、防火墙等的控制不在本文讨论范围内。它们之中任何一点拿出来都够写一本书了。

首先，要在部署服务器可以访问到的地方创建一个新的版本库。

读者可以在公司/实验室里搭建 GitLab，或者连接到 GitHub 上。如果是自己搭建 GitLab，其实完全可以把 Git 服务器和生产服务器放在同一台机器上。可以通过以下命建立 Server 端的远程版本库，也可以通过访问 Git 服务器的 Web 界面自行操作。

```
$ mkdir -p ~/git/project.git
$ cd ~/git/project.git
$ git --bare init
```

值得注意的是，即使是把 Git Server 和 Web Server 放在同一台机器上，Capistrano 也是通过 Web 服务的形式访问 Git。这里可以配置公钥来简化整个流程。

```
$ test -e ~/.ssh/id_dsa.pub || ssh-keygen -t dsa
$ cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
```

还有一件非常重要的事情：Capistrano 将在应用程序目录名和 Rails 子目录中插入名为 current 的目录，其中包括 public 子目录。在用 Capistrano 部署好的项目中，目录结构类似下面的样子（current 是当前部署的版本，releases 是该分支下根据版本生成的曾经的项目，shared 中是各个版本共享的资源）：

```
$ ls project
> current    releases    shared
```

而这个 public 目录就是我们熟悉的 Rails 程序中静态资源目录；也就是说，不论你用 Apache 或 Ngnix 还是其他 Web 服务器来部署静态资源，需要把这个目录设置为根目录。以 Apache 配置为例：

```
DocumentRoot /home/project/current/public/
```

到此为止，对服务器的预先配置就已经完成了，接下来的工作都可以在开发环境中完成。

对应用进行版本控制

最开始，要在项目的 Gemfile 中加入 Capistrano gem：

...

gem 'capistrano'

...

然后执行命令 bundle install 来安装 gem。接下来执行一条很重要的命令，它的作用是把项目中用到的 gem 及其版本号都记录下来：

$ bundle pack

//然后可以做一个版本记录，并推送到远程库

$ git add .

$ git commit -m "install capistrano & bundle gems"

$ git push origin master

以上，已经完成了项目 gem 整理。接下来我们要把项目代码放入生产环境。

远程部署应用程序

接下来的过程，我们可以先猜测一下：

把代码 pull 到服务器的 DocumentRoot

在生产服务器上生成数据库

安装 gem

重启 Passenger

……

这些事情，可以想到的话，就可以手动来做；但是等下，要 Capistrano 是做什么的……

没错，到了这一步，就可以显示 Capistrano 的威力了：自动化部署。以上的事情可以全交给它来做，当然前提是要写一份自动化部署的脚本。

首先，在开发环境的项目根目录中执行命令：

$ capify

[add] writing './Capfile'

[add] writing './config/deploy.rb'

[done] capified!

//Capistrano3 中使用下面的命令

$ cap install

现在多出来了两个文件：Capfile 和 deploy.rb，前者是 Capistrano 对 Rakefile 的模拟，不需要改动，后者是一个自动化的脚本，需要编辑的就是这个。这个文件里需要用到一些 Capistrano 的 DSL，可以看这里参考。在此给出一份比较简单的脚本作为参考，它是上面命令生成的默认脚本。

```
# config valid only for Capistrano 3.1
lock '3.1.0'
set :application, 'my_app_name'
set :repo_url, 'git@example.com:me/my_repo.git'
namespace :deploy do
  desc 'Restart application'
  task :restart do
    on roles(:app), in: :sequence, wait: 5 do
      # Your restart mechanism here, for example:
      # execute :touch, release_path.join('tmp/restart.txt')
    end
  end
  after :publishing, :restart
  after :restart, :clear_cache do
    on roles(:web), in: :groups, limit: 3, wait: 10 do
      # Here we can do anything such as:
      # within release_path do
      #      execute :rake, 'cache:clear'
      # end
  end
 end
```

end

在修改好:application、:repo_url 等值后，就可以开始进行部署了。

第一次部署的时候，请运行：

$ cap deploy:setup

之后会要求输入服务器密码。连接成功后会在生产服务器上生成必要的目录。

如果想测试配置，看还有没有别的问题，可以运行：

$ cap deploy:check

一旦解决完所有问题，就可以运行下面的命令，进行数据库生成，并完成部署：

$ cap deploy:migrate

$ cap deploy

至此，就已经把项目部署到生产服务器上了。

版本控制推进部署

已经部署完了么？

并不是这样的，不要忽视掉最开始强调版本控制的用意。Capistrano 能让你在持续开发的情况下进行版本推进部署，而操作非常简单。每次做好提交、并把版本库推送到远端后，就可以运行 cap deploy 进行部署。

如果有某些原因，需要回到上一个部署的版本，执行命令：

$ cap deploy:rollback

这就完了？

没错！整个部署的过程就是这样。

如果想掌握关于它的高级用法，可以看 Wiki 文档。

原文链接：http://blog.segmentfault.com/pengedy/1190000000409329

Java IO vs NIO vs AIO vs 协议 Servlet 3.0 以及 NIO 的框架

nio（new io）

同步 ： 自己亲自出马持银行卡到银行取钱（使用同步 IO 时，Java 自己处理 IO 读写）。

异步 ： 委托一小弟拿银行卡到银行取钱，然后给你（使用异步 IO 时，Java 将 IO 读写委托给 OS 处理，需要将数据缓冲区地址和大小传给 OS（银行卡和密码），OS 需要支持异步 IO 操作 API）。

阻塞 ： ATM 排队取款，你只能等待（使用阻塞 IO 时，Java 调用会一直阻塞到读写完成才返回）。

非 阻塞 ： 柜台取款，取个号，然后坐在椅子上做其它事，等号广播会通知你办理，没到号你就不能去，你可以不断问大堂经理排到了没有，大堂经理如果说还没到你就不能去 （使用非阻塞 IO 时，如果不能读写 Java 调用会马上返回，当 IO 事件分发器会通知可读写时再继续进行读写，不断循环直到读写完成）。

一、IO NIO AIO

io、nio、aio 的区别，类似于 resin、apache、nginx 在 io 处理上的区别，从多线程互不干扰的阻塞式执行（resin），到轮询式的同步非阻塞式（apache），再到异步非阻塞式（nginx）。

现在这三种 io 都在 jdk 中予以了支持。

1、IO （BIO）

同步并阻塞，服务器实现模式为一个连接一个线程，每个线程亲自处理 io 并且一直等待 io 的完成，即客户端有连接请求时服务器端就需要启动一个线程进行处理，如果这个连接不做任何事情会造成不必要的线程开销，当然可以通过线程池机制改善。

IO 的局限：IO 是面向流的，阻塞式的，串行的一个过程。对每一个客户端的 socket 连接，IO 都需要一个线程来处理，而且在此期间，这个线程一直被占用，直到 socket 关闭。在这期间，tcp 的连接、数据的读取、数据的返回都是被阻塞的。也就是说这期间大量的浪费了 cpu 的时间片和线程占用的内存资源。

每建立一个 Socket 连接时，同时创建一个新线程对该 Socket 进行单独通信（采

用阻塞的方式通信)。这种方式具有很高的响应速度，并且控制起来也很 简单，在连接数较少的时候非常有效，但是如果对每一个连接都产生一个线程的无疑是对系统资源的一种浪费，如果连接数较多将会出现资源不足的情况。

2、NIO （new IO） 从 jdk1.4 开始

同步非阻塞，服务器实现模式为一个请求一个线程，每个线程亲自处理 io，但有另外的线程轮询检查是否 io 准备完毕，不必等待 io 完成，即客户端发送的连接请求都会注册到多路复用器上，多路复用器轮询到连接有 I/O 请求时才启动一个线程进行处理。

NIO 则是面向缓冲区的，非阻塞式的，基于选择器的，用一个线程来轮询监控多个数据传输通道，哪个通道准备好了（即有了一组可以处理的数据），就处理哪个通道。

服务器端保存一个 Socket 连接列表，然后对这个列表进行轮询，如果发现某个 Socket 端口上有数据可读时（读就绪），则调用该 socket 连接的相 应读操作；如果发现某个 Socket 端口上有数据可写时（写就绪），则调用该 socket 连接的相应写操作；如果某个端口的 Socket 连接已经中断，则调用相应的析构方法关闭 该端口。这样能充分利用服务器资源，效率得到了很大提高。

3、AIO (Asynchronous io、NIO.2) 从 jdk1.7 开始

http://stevex.blog.51cto.com/4300375/1284437

异步非阻塞，服务器实现模式为一个有效请求一个线程，客户端的 I/O 请求都是由 OS 先完成了再通知服务器应用去启动线程进行处理，每个线程不必亲自处理 io，而是委派 os 来处理，并且也不需要等待 io 完成了，如果完成后，os 会通知的。

采用 linux 的 epoll 模型。

3、结论

在连接数不多的情况下，传统 IO 编写容易、方便使用。但是随着连接数的增多，问题传统 IO 就不行了。因为传统 IO 处理每个连接都要消耗一个线程，而程序的效率当线程数不多时是随着线程数的增加而增加，但是到一定的数量之后，是随着线程数的增加而减少。所以传统阻塞式 IO 的 瓶颈在于不能处理过多的连接。

非阻塞式 IO 的出现的目的就是为了解决这个瓶颈。而非阻塞式 IO 是怎么实现的呢？非阻塞 IO 处理连接的线程数和连接数没有联系，也就是说处理 10000 个连接非阻塞 IO 不需要 10000 个线程，你可以用 1000 个也可以用 2000 个线程来处理。因为非阻塞 IO 处理连接是异步的。当某个连接发送请求到服务 器，服务器把这个连接请求当作一个请求"事件"，并把这个"事件"分配给相应的函数处理。我们可以把这个处理函数放到线程中去执行，执行完就把线程归还。 这样一个线程就可以异步的处理多个事件。而阻塞式 IO 的线程的大部分时间都浪费在等待请求上了。

然后 NIO 的非阻塞，需要一直轮询，也是一个比较耗资源的。所以出现 AIO

4、BIO、NIO、AIO 适用场景

BIO 方式适用于连接数目比较小且固定的架构，这种方式对服务器资源要求比较高，并发局限于应用中，JDK1.4 以前的唯一选择，但程序直观简单易理解。

NIO 方式适用于连接数目多且连接比较短（轻操作）的架构，比如聊天服务器，并发局限于应用中，编程比较复杂，JDK1.4 开始支持。

AIO 方式使用于连接数目多且连接比较长（重操作）的架构，比如相册服务器，充分调用 OS 参与并发操作，编程比较复杂，JDK7 开始支持。

ps：AIO、NIO 是基于 IO 的，并不是取代 IO 的意思。

二、NIO 的框架

IO 的概念：

1. 就 IO 而言：概念上有 5 中模型：blocking I/O, nonblocking I/O, I/O multiplexing (select and poll), signal driven I/O (SIGIO), asynchronous I/O (the POSIX aio_functions)。

2. 然后呢 不同的操作系统对上述模型支持不同: unix 支持 io 多路复用，不同系统叫法不同 :freebsd 里面叫 kqueue; linux 是 epoll。而 windows: 2000 的时候就诞生了 IOCP 支持最后一种异步 I/O

3. java 是一种跨平台语言，为了支持异步 IO, 诞生了 nio, Java1.4 引入的 NIO 1.0 是基于 I/O 复用的。在各个平台上会选择不同的复用方式。Linux 用的 epoll, BSD 上用 kqueue, Windows 上应该是重叠 I/O（肯定不是 IOCP)。

4:基于 jdk 的 nio ,不同公司出了一堆框架：apache mina ， jboss 的 netty，sun 的 grizzly。 这些都是直接封装传输层的 tcp/udp。

nio 直接使用比较难用,所以有了 netty 等针对网络 io 部分(tcp/udp-传输层)的封装（nio 也有非网络 io 部分)，为了使 nio 更易用。 netty 等只是一个 nio 框架，不需要 web 容器的额外支持，也就是说不限定 web 容器。

3 个 NIO 框架是：

1、Mina

Mina(Multipurpose Infrastructure for Network Applications) 是 Apache 组织一个较新的项目，它为开发高性能和高可用性的网络应用程序提供了非常便利的框架。当前发行的 Mina 版本 2.04 支持基于 JavaNIO 技术的 TCP/UDP 应用程序开发、串口通讯程序,Mina 所支持的功能也在进一步的扩展中。目前，正在使用 Mina 的应用包括：Apache Directory Project、AsyncWeb、AMQP（Advanced MessageQueuing Protocol)、RED5 Server (Macromedia? FlashMedia RTMP)、ObjectRADIUS、 Openfire 等等。

2、Netty

Netty 是一款异步的事件驱动的网络应用框架和工具，用于快速开发可维护的高性能、高扩展性协议服务器和客户端。也就是说，Netty 是一个 NIO 客户端/服务器框架，支持快速、简单地开发网络应用，如协议服务器和客户端。它极大简化了网络编程，如 TCP 和 UDP 套接字服务器。

3、Grizzly

Grizzly 是一种应用程序框架，专门解决编写成千上万用户访问服务器时候产生的各种问题。使用 JAVANIO 作为基础，并隐藏其编程的复杂性。容易使用的高性能的 API。带来非阻塞 socketd 到协议处理层。利用高性能的缓冲和缓冲管理使用高性能的线程池。

三、Servlet3.0 vs NIO

servlet3.0 是一个规范、或者协议，可以用 IO 实现，也可以用 NIO 实现，而 NIO 则只是一种技术实现。类似于一个是架构，一个是具体技术。

相同：

都提供了异步功能。

不同：

jdk 的 nio 直接使用比较难用，所以有了 netty 这些针对网络 io 部分（tcp/udp-传输层）的封装(nio 也有非网络 io 部分),为了使 nio 更易用而已。servlet3.0 是另外一个东西，不是对 io 的封装，而是 javaee6 众多规范中的一个。但凡 javaee6 的实现（或者像 tomcat 这种 web 容 器部分的实现），都会支持 servlet3.0，servlet 理论上可以支持多种应用层协议（不单单只是 http），而 servlet3.0 以后提供的 异步特性与 javase 提供的 nio 或 aio 无直接关系，就是使用 bio 一样可以实现 servlet3.0 中提供的异步特性。

原文链接：http://qindongliang1922.iteye.com/blog/2017278

Integer.valueOf(String) 方法之惑

Importnew 注：如果你也对 Java 技术翻译分享感兴趣，欢迎加入我们的 Java 开发小组。参与方式请查看小组简介。

有个仁兄在 StackOverflow 上发起了一个问题，是这么问的：

“ 我被下面的代码搞晕了，为什么它们会返回不同的值？”

```
1	System.out.println(Integer.valueOf("127")==Integer.valueOf("127"));
2	System.out.println(Integer.valueOf("128")==Integer.valueOf("128"));
3	System.out.println(Integer.parseInt("128")==Integer.valueOf("128"));
```

输出是：

```
1	true
2	false
3	true
```

为什么第一个判断返回了 true 而第二个判断返回了 false？127 和 128 有什么我不知道的区别吗？（当然除了 127 小于 128…）

还有，为什么第三个判断返回了 true？

我看了另一个相关提问的回答，但是我还是不知道它们什么时候返回 true 还有为什么第二个判断返回 false。

回答#1：

Integer.valueOf(String)确有一个不同寻常的行为。

valueOf 会返回一个 Integer（整型）对象，当被处理的字符串在-128 和 127（包含边界）之间时，返回的对象是预先缓存的。这就是为什么第一行的调用会返回 true-127 这个整型对象是被缓存的(所以两次 valueOf 返回的是同一个对象)——第二行的调用返回 false 是因为 128 没有被缓存，所以每次调用，都会生成一个新的整型对象，因此两个 128 整型对象是不同的对象。

重要的是你要知道在上面的比较中，你实际进行比较的是 integer.valueOf 返回的对象引用，所以当你比较缓存外的整型对象时，相等的判断不会返回 true，就算你传个 valueOf 的值是相等的也没用。(就像第二行中 Integer.valueOf(128)==Integer.valueOf(128))。想让这个判断返回 true，你需要使用 equals()方法。

parseInt()返回的不是整型对象，而是一个 int 型基础元素。这就是为什么最后一个判断会返回 true，第三行的判断中，在判断相等时，实际比较的是 128 == 128，所以它必然是相等的。

再来说说第三种比较中的一点区别，使得它的结果与第二种比较不一样了：

一个 unboxing conversion（一种比较时的转换，把对对象的引用转换为其对应的原子类型）在第三行的比较中发生了。因为比较操作符使用了==同时等号的两边存在一个 int 型和一个 Integer 对象的引用。这样的话，等号右边返回的 Integer 对象被进一步转换成了 int 数值，才与左边进行相等判断。

所以在转换完成后，你实际比较的是两个原子整型数值。这种转换正是你在比较两个原子类型时所期待看到的那样，所以你最终比较了 128 等于 128。

回答#2：

Integer 类有一个静态缓存，存储了 256 个特殊的 Integer 对象——每个对象分别对应`-128 和 127 之间的一个值。

有了这个概念，就可以知道上面三行代码之间的区别。

```
1        new Integer(123);
```

显示创建了一个新的 Integer 对象。

```
1        Integer.parseInt("123");
```

解析完字符串后返回一个 int 值。

```
1        Integer.valueOf("123");
```

这种情况比其他的要更复杂一些。首先进行了字符串解析，然后如果解析的值位于-128 和 127 之间，就会从静态缓存中返回对象。如果超出了这个范围，就会调用 Integer()方法并将解析的值作为参数传入，得到一个新的对象。

现在，让我们看一下问题中的 3 个表达式。

```
1        Integer.valueOf("127")==Integer.valueOf("127");
```

上面的表达式返回 true，因为 Integer 的值从静态缓存中取了 2 次，表达式返回了对象与自己比较的结果。因为只有一个 Integer 对象，所以返回结果为 true。

```
1        Integer.valueOf("128")==Integer.valueOf("128");
```

上面的表达式返回 false，因为 128 没有存在静态缓冲区。所以每次在判断相等时等式两边都会创建新的 Integer 对象。由于两个 Integer 对象不同，所以==只有等式两边代表同一个对象时才会返回 true。因此，上面的等式返回 false。

```
1        Integer.parseInt("128")==Integer.valueOf("128");
```

上面的表达式比较的是左边的原始 int 值 128 与右边新创建的 Integer 对象。但是因为 int 和 Integer 之间比较是没有意义的，所以 Java 在进行比较前会将 Integer 自动拆箱，所以最后进行的是 int 和 int 值之间的比较。由于 128 和自己相等，所以返回 true。

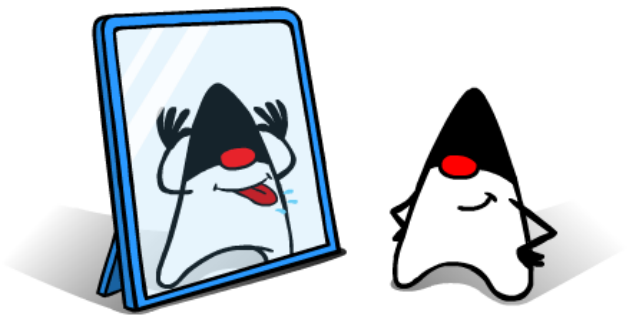
补充说明:文章没有列出具体的执行环境,jdk 版本信息,因此造成不一样的结果,让大家误解了.抱歉.现在说明一下:

此文只适应于 jdk7 或以上版本,因为 jdk6 与 jdk7 的 Integer 具体实现有差别,详情可查看下源代码.

原文链接：http://www.importnew.com/9162.html

在运行期通过反射了解 JVM 内部机制

在日常工作中，我们都习惯直接使用或者通过框架使用反射。在没有反射相关硬编码知识的情况下，这是 Java 和 Scala 编程中使用的类库与我们的代码之间进行交互的一种主要手段。但是，使用反射仅限于 JVM 内部运行的 Java 和 Scala 代码。假使在运行期通过反射既能查看自己的代码又能看到 JVM 的代码，会有怎样的效果呢 ？



当我们开始创建 Takipi 的时候，试图寻找一种通过分析 JVM 堆内存来进行一些底层优化的有效方法，比如扫描一个托管堆块（managed heap block）的地址空间。我们找到了许多有趣的工具和组件用来检测 JVM 状态的各个方面，其中一个就是在运行期通过反射了解 JVM 内部机制。

译注： Takipi 是一家以色列创业公司，发现了一种构建后端服务器网络的新方式，可以使用非常简化的除错流程帮助企业定位错误。

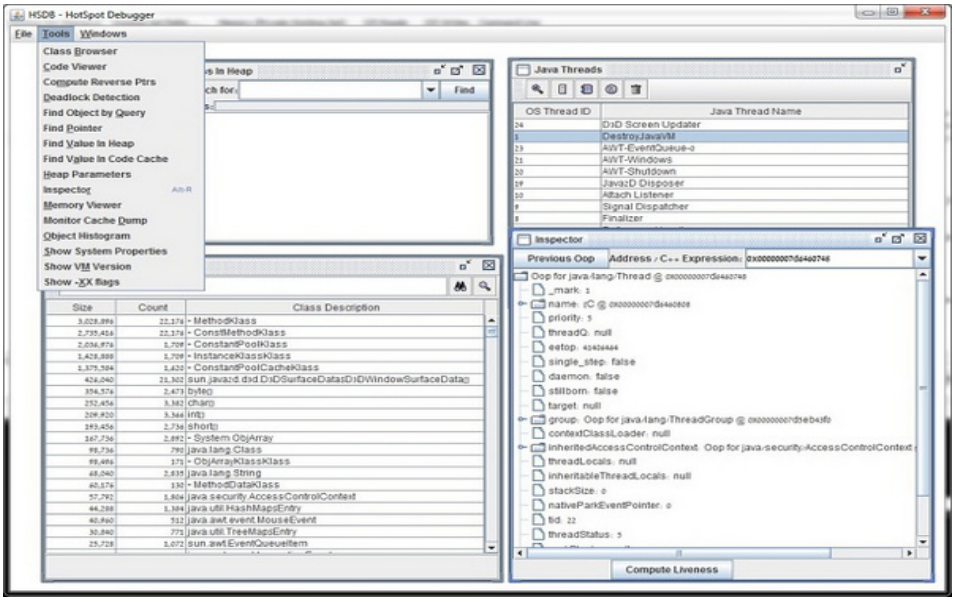
Java 可服务性代理（Serviceablity Agent，简写 SA）是最强大和最底层的 Java 调试工具之一。这个强大的工具是 HotSpot JDK 自带的。使用它不仅可以看到堆中的 Java 对象，还可以看到内部 C++ 对象，包括 JVM 本身。那才是真正魔法开始的地方。

反射的构成：在运行时动态检测和修改对象时，无论使用何种形式的反射都不能缺少两个必要信息。第一个是想要检测的对象引用（或者地址）；第二个是对象结构描述，包括所有字段的偏移量以及它们的类型信息。如果支持动态方法调用，这个结构还需要包括类方法表 （比如 vtable）的引用，以及每个方法需要的参数。

Java 反射本身是非常简单的，通过反射获取目标对象的引用与获取其它方式一样。使用 Object.getClass 通用方法（最开始从类的字节码中加载）获取字段和方法结构。真正的问题是：如何反射 JVM 本身？

反射的关键：令人惊奇的是， JVM 通过一套公开的输出符号暴露了其内部的类型系统。 这些符号给可服务性代理（或其他工具）提供了访问 JVM 内部类系统结构和地址的方法。通过这些符号，几乎可以检测到 JVM 内部在最底层运行机制的所有方面，包括诸如原始堆地址、线程/栈地址以及编译器内部状态等。

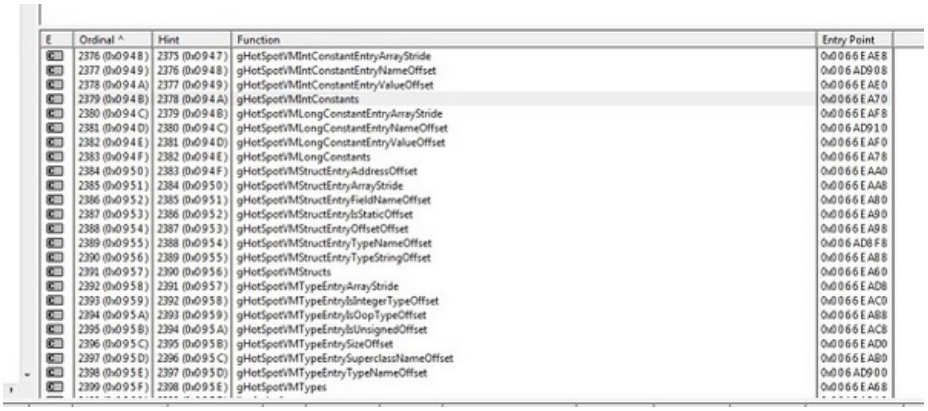
反射实战：为了增加对这种方式的了解，启动可服务性代理的 HotSpot 调试程序界面，可以看到正在运行的一些功能。将 sun.jvm.hotspot.HSDB 作为主类参数启动 sa-jdi.jar，可以看到与其它 JVM 功能强大调试工具，例如 jmap、jinfo 和 jstack 等，一样的底层信息。



HSDB 及其为目标 JVM 提供的一些非常底层的检测功能具体实现：让我们仔细了解这些由 JVM 提供的功能。这种方法的基础是由 jvm 函数库公开导出的 gHotSpotVMStructs 结构。这个结构暴露了 JVM 内部的类型系统，也为我们提供了可以开始反射的根对象地址。可以通过 JNI 或者 JNA 访问这个符号，调用方式与访问动态链接的操作系统公开系统库符号一样。

接下来问题就变成：怎样解析由 gHotSpotVMStructs 符号中地址的数据？从下表中可以看到，JVM 不仅暴露了类型系统的地址以及根对象地址，还提供一些额外的符号用来解析需要的数据值。这些数据包括已定位的类描述符及其二进制

偏移量。



E	Ordinal ^	Hint	Function	Entry Point
	2376 (0x0948)	2375 (0x0947)	gHotSpotVMIntConstantEntryArrayStride	0x0066EAE8
	2377 (0x0949)	2376 (0x0948)	gHotSpotVMIntConstantEntryNameOffset	0x006AD908
	2378 (0x094A)	2377 (0x0949)	gHotSpotVMIntConstantEntryValueOffset	0x0066EAE0
	2379 (0x094B)	2378 (0x094A)	gHotSpotVMIntConstants	0x0066EA70
	2380 (0x094C)	2379 (0x094B)	gHotSpotVMLongConstantEntryArrayStride	0x0066EAF8
	2381 (0x094D)	2380 (0x094C)	gHotSpotVMLongConstantEntryNameOffset	0x006AD910
	2382 (0x094E)	2381 (0x094D)	gHotSpotVMLongConstantEntryValueOffset	0x0066EAF0
	2383 (0x094F)	2382 (0x094E)	gHotSpotVMLongConstants	0x0066EA78
	2384 (0x0950)	2383 (0x094F)	gHotSpotVMStructEntryAddressOffset	0x0066EAA0
	2385 (0x0951)	2384 (0x0950)	gHotSpotVMStructEntryArrayStride	0x0066EAA8
	2386 (0x0952)	2385 (0x0951)	gHotSpotVMStructEntryFieldNameOffset	0x0066EA80
	2387 (0x0953)	2386 (0x0952)	gHotSpotVMStructEntryIsStaticOffset	0x0066EA90
	2388 (0x0954)	2387 (0x0953)	gHotSpotVMStructEntryOffsetOffset	0x0066EA98
	2389 (0x0955)	2388 (0x0954)	gHotSpotVMStructEntryTypeNameOffset	0x006AD8F8
	2390 (0x0956)	2389 (0x0955)	gHotSpotVMStructEntryTypeStringOffset	0x0066EA88
	2391 (0x0957)	2390 (0x0956)	gHotSpotVMStructs	0x0066EA60
	2392 (0x0958)	2391 (0x0957)	gHotSpotVMTypeEntryArrayStride	0x0066EAD8
	2393 (0x0959)	2392 (0x0958)	gHotSpotVMTypeEntryIsIntegerTypeOffset	0x0066EAC0
	2394 (0x095A)	2393 (0x0959)	gHotSpotVMTypeEntryIsOopTypeOffset	0x0066EAB8
	2395 (0x095B)	2394 (0x095A)	gHotSpotVMTypeEntryIsUnsignedOffset	0x0066EAC8
	2396 (0x095C)	2395 (0x095B)	gHotSpotVMTypeEntrySizeOffset	0x0066EAD0
	2397 (0x095D)	2396 (0x095C)	gHotSpotVMTypeEntrySuperclassNameOffset	0x0066EAB0
	2398 (0x095E)	2397 (0x095D)	gHotSpotVMTypeEntryTypeNameOffset	0x006AD900
	2399 (0x095F)	2398 (0x095E)	gHotSpotVMTypes	0x0066EA68

错误!

jvm.dll 暴露符号的一个 dependency walker 截图清单（manifest）：

gHotSpotVMStructs 结构指向了类及其字段的一个列表，每个类提供了一个字段列表。对于每个字段，该结构提供了它的名称、类型以及是否静态字段。如果是一个静态字段，这个结构还会提供目标对象的访问地址。该地址可用作反射 JVM 内部特定组件的根，包括诸如编译器、线程处理或者收集堆系统等。

可以从这里检出 Hotspot JDK 中可服务性代理用来解析 gHotSpotVMStructs 的实际算法。

实例：既然已经大概了解了这些功能，现在看一些由该接口访问数据类型的示例。那些编写 SA 代理的人，在为 gHotSpotVMStructs 表中的类创建 Java 封装时克服了很多麻烦。在访问大多数内部系统时，这些封装提供一套简洁的 API，不仅类型安全而且还隐藏了一些解析数据所必须的二进制工作。

为了更好地了解这些 API 提供的强大功能，下面是其中的一些底层类介绍：

VM 是一个单例类，它暴露了 JVM 的许多内部系统，比如线程系统、内存管理以及集合功能。可以把它当作 JVM 许多子系统的一个入口。当你研究 API 时，可以从这里入手。

JavaThread 可以让你从 JVM 角度理解一个 Java 线程，同时还有（编译后的、经过解释的和本地）帧位置和类型的深入信息以及实际的本地堆栈和 CPU 寄存器信息。

CollectedHeap 可以让你研究收集堆（collected heap）的原始内容。由于 HotSpot 包含多种 GC 的实现，CollectedHeap 就是供具体实现比如 ParallelScavengeHeap 必须继承的抽象类。每一个都提供包含 Java 对象实际地址的一组内存区域。

当你在看每个类的实现时，你会发现本质上它们就是使用类似反射 API 来查看 JVM 内存硬编码的包装器。

C++中的反射：这些 Java 包装器中的每一个都被设计为 JVM 内部一个近乎完整的 C++类镜像。我们知道，C++ 没有原生的反射能力，这就让我们产生了一个疑问：如何在这之间建立的纽带？

答案就是 JVM 开发者所了一些非常特别的事情。经过一系列 C++宏指令以及大量的艰苦工作，HotSpot 开发小组手动将许多内部 C++类的字段结构映射并加载到全局的 gHotSpotVMStructs 中。这个过程就是可以从外部进行反射的原因。在 JVM 编译期间确定了实际字段偏移量的值和布局，帮助并确保了输出结构兼容 JVM 目标操作系统。

跨进程连接：可服务性代理有另外一个强大的功能值得我们关注。从进程外反射一个内部运行的 JVM 是 SA 框架提供的超酷功能之一，通过把可服务性代理附加到目标 JVM 作为一个操作系统级别的调试器可以实现这一功能。由于 SA 代理框架是依赖于操作系统的，对于 Linux 会利用 gdb 调试器进行连接，对于 Windows 则会使用 winDbg （这意味着需要 Windows 调试工具）。调试器框架是可扩展的，这意味着可以通过继承 DebuggerBase 这个抽象类来使用另一个调试器。

一旦建立了调试器的连接，gHotSpotVMStruct 的返回地址值会传回到调试器进程，这样可以（借助操作系统）开始检测甚至是修改目标 JVM 的内部对象系统。这就是 HSDB 如何实现连接并调试一个目标 JVM，包括 Java 代码和 JVM 代码。 **错误!**

HSDB 接口为 SA 代理提供了反射目标 JVM 进程功能希望本文能够勾起你的兴趣。 以我个人观点来看，这个架构是我最喜欢的 JVM 设计之一 。它的优雅和开放绝对令人惊叹。在我们构建 Takipi 实时编码模块时超级有用， 所以要感谢它的设计者。

你是否已经在代码中使用了这些 API？非常乐意在下面的评论中听到它，或者回答你们的任何问题。

原文链接：http://segmentfault.com/a/1190000000406956

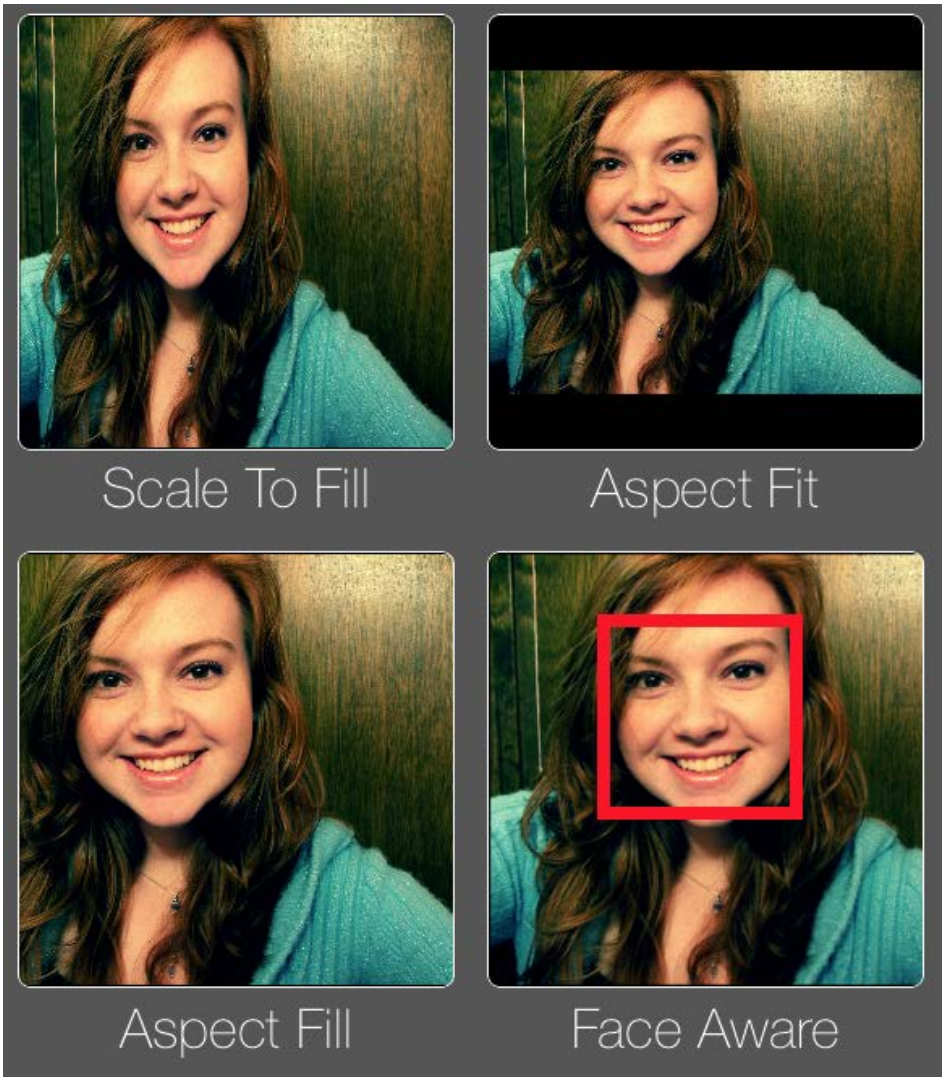
Objective-C 相关 Category 的收集

Categories 是给你得不到源码的 classes 增加功能的一种方法。这个页面收集一些相关的 Category，并且持续更新，你可以订阅关注。作者是 Fille Åström，是 @ IMGNRY 的联合创始人和开发者。

UIImageView+FaceAwareFill

这个类别使用了 Aspect Fill 内容模式，可以自动根据图像内容进行调整，当检测到人脸时，它会以脸部中心替代掉以图片的几何中心。

测试环境：Xcode 5.0，iOS 6.0 以上



NSRegularEx+ObjCRegex

Objective-C-RegEx-Categories 是 NSRegularExpression 的一个延展，它可以把 Object-C 中的很多正则表达式合并成一个，简化了代码。

这个库没有任何依赖性，适用于 iOS 4+和 OS X 10.7+。



NSObject+AutoCoding

AutoCoding 是一个 NSObject 的类目，提供了对 NSCoding 和 NSCopying 的自动支持。

兼容 ARC 和 non-ARC 编译目标

支持 iOS 7.0/Mac OS 10.9 (Xcode 5.0, Apple LLVM compiler 5.0)

NSInvocation+SimpleCreation

创建 invocations 的简单方法

NSString+SHPersian

SHPersian 是一个针对使用波斯语和阿拉伯语的 iOS 开发者提供的工具，包含了一个在 Persian iOS app 中为文本添加自定义外观的必需的类。

UILabel+ContentSize

在 UILabel 内计算内容的大小

UIViewController+Swizzled

记录 UIViewController 层次：包括你在视图控制器的名字，还有你进入层次的展示。

NSObject+Association

你是不是一直希望将"userInfo"显示到 UIAlertView 上呢？通过 Association 这

个 category 可以将任意的对象赋值给其它任意对象(从 iOS3.1 和 mac os 10.6 到最新的系统版本。)

NSObject+Association.m

```
//
//  NSObject+Association.m
//
//  Created by Maciej Swic on 03/12
//  Released under the MIT License.
//

#import <objc/runtime.h>

#import "NSObject+Association.h"

@implementation NSObject (Associati

static char associatedObjectsKey;

- (id)associatedObjectForKey:(NSStr
  NSMutableDictionary *dict = objc_
  return [dict objectForKey:key];
}

- (void)setAssociatedObject:(id)obj
  NSMutableDictionary *dict = objc_
  if (!dict) {
      dict = [[NSMutableDictionary
```

NSHTTPCookieStorage+FreezeDry

app 重启时，清除 UIWebView cookies。

UIDevice+Hardware

检测硬件设备的版本。

NSObject+LogDealloc

是一个 NSObject category，对于内存泄露的跟踪非常有用

NSObject+SFObservers

是对 NSNotificationCenter and KVO 的一个扩展，它能够自动移除观察者。

UIApplication+NetworkActivity

UIApplication+NetworkActivity 跟踪你最近进行过的网络操作，并管理 NetworkActivityIndicator。

UIView+Recursion

以递归的方式遍历(查找)subview

UIView+RoundedCorners

使用图层蒙版为视图添加圆角

UIView+Stacker

Stack subviews 是按照索引进行垂直排序的。主要用于——使用 xib 时，以及需要进行 view 布局时(显示/隐藏)——当基于外部数据。不过不能用于 autolayout。.

NSString+PDRegex

简化正则表达式的使用

MKMapView+MoveLogo

这个类目允许你移动 MKMapView logo，即使你放其他东西在 mapview 上它仍能保持可见。如果隐藏了 logo，那将不能通过 App Store 审核。已经在 iOS 5-iOS 7 上进行了测试。

NSDate+Escort

NSDate-Escort 是一个 NSDate 实用库，兼容 NSDate-Extensions API

UILabel+DynamicSizeMe

调整 UILabel 来根据内容改变其框架

UIView+Toast

适用于 iOS 上的 Android 风格 toas 通知。

UIImage+NYXImagesKit

NYXImagesKit 是一个重组了多个有用的 UIImage categories 的 iOS 项目，可对图像/图片进行多个处理，比如筛选、模糊、优化、蒙版、调整大小、旋转以及保存等等。同时还提供了一个 UIImageView 子类从 URL 异步加载图片，并在下载完毕时展示图片。

UIViewController+MJPopup

MJPopupViewController 是一个 UIViewController Category，用于使用不同的过渡效果来把 ViewController 作为弹出视图进行展示。

UIImage+MGImageUtilities

MGImageUtilities 展示两个 UIImage category：UIImage+ProportionalFill 和 UIImage+Tint。你可以通过 UIImage+ProportionalFill 调整任意图片的尺寸，可以

使用 UIImage+Tint 来为图片着色。

MultipleObjects+DTFoundation

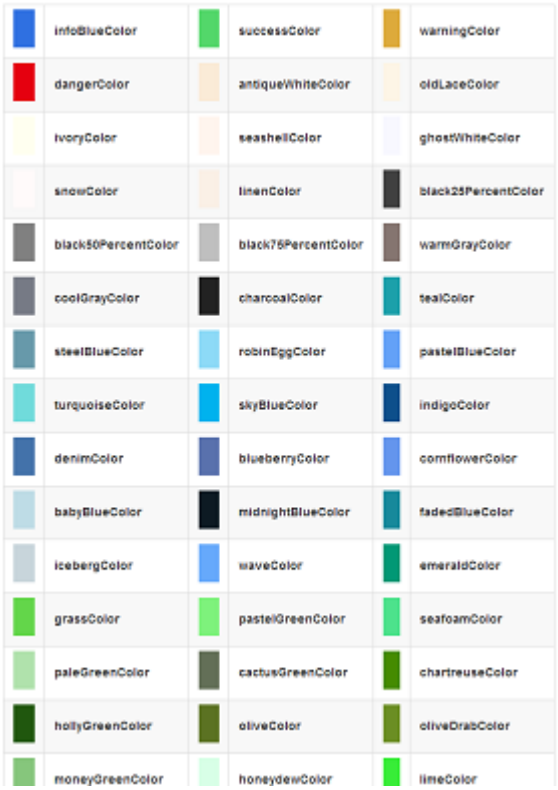
DTFoundation 集合了实用方法和 category 的扩展，逐渐演变成一个文档齐全的工具集，记录和测试代码以加快开发。

UIImageView+WebCache

SDWebImage 提供一个 UIImageView 类别，以支持加载管理源自网络的远程图片。具有异步加载、缓存管理、同一个 URL 下载次数控制和优化等特征。简单易用。

UIColor+Colours

100 组漂亮的预制的色彩和配色方案可以让你的 iOS/OS X 开发更轻松。



UIImage+PDFColoredImage

UIImage-PDFColoredImage 是一个 UIImage 扩展，可以把黑色的 PDF image 转换成任意尺寸任意颜色的 UIImage，只需一行代码，简单易用。

NSData+AsyncCacher

NSData-AsyncCacher 是 NSData 的一个 category，用于从 url 和 block 中异步加载数据。请求数据使用 NSCache 缓存，可以多次请求。

NSDate+Helper

通过一些便捷函数扩展了 Cocoa 的 NSDate

MultipleObjects+ObjectiveSugar

Objective-C additions for humans. Ruby style.



MKMapView+ZoomLevel

在 MapView 中设置缩放比例。

NSString+Extensions

一个针对 NSStrings 的 category，它能将 string 转换为 SHA1，NSNumber。检测 string 是否为空、是否包含某个子字符串，以及替换掉 NSDictionary 中的子字符串。

UIView+Utils

ViewUtils 是一个 category methods 的集合，这些方法是 UIView 的扩展，包含了许多方便的属性和方法，其中许多都是我们希望苹果内置的。

NSDate+TimeAgo

让 NSDate 报告时间，比如"A moment ago", "30 seconds ago", "5 minutes ago", "Yesterday", "Last month", "2 years ago"以及其他。

NSString+FontAwesome

FontAwesome+iOS 是一个开源形象字体库，通过扩展 NSString 让你在 app 中轻松使用 FontAwesome 字体。

NSArray+FirstObject

从数组中获得第一个对象，或者从空数组中返回零

NSObject+AutoDescription

能够简单地为任何类描述日志(从 NSObject 派生出来)

CGRect+Additions

处理 CGRects 的一些函数。

UIView+ModifyFrame

简单实用的 UIView 类目使修改框架不那么变态。

NSString+UsefulShit

Objective-C category 示范

原文链接：http://www.cocoachina.com/applenews/devnews/2014/0212/7808.html

豆瓣 CODE 两年历程回顾：git 不是万能的，没有 review 是万万不能的

2014 年 2 月 14 日是豆瓣 CODE 项目成立两周年，同日豆瓣 CODE 团队宣布将整个框架开源。本次开源包含 CODE 框架以及 Mikoto、Linguist、P、CodeLive 等周边项目，感兴趣的工程师可以在这个基础上继续开发。

下面，豆瓣 CODE 的工程师们将在 InfoQ 上分享一些 CODE 背后的故事。

背景

2012 年初，已经对 SVN 和 Mercurial 无力吐槽的豆瓣工程师们正面临一个艰难的决定：是往 Github 企业版迁移，还是自己研发一套代码管理系统。

CODE —— C: Community O: Original D: Developer E: Eldamar

2012 年 2 月 14 日，清风老师为豆瓣 CODE 写入了第一行代码；一周之后，一个用 Trac 改造的原型诞生了。CODE 不是一个公司项目，自始至终没有任何一个豆瓣的产品团队或工程团队为这套系统负责，但一个又一个的豆瓣工程师开始被吸引到这个项目周围，形成了一个强大的虚拟团队。

随着 Github 在国内的普及程度越来越高，越来越多的企业也开始探索自建 git 仓库之路。开源的 Gitlab 项目开始受到部分团队的关注，但这个项目仍有较多局限性。这时，豆瓣 CODE 引起了包括淘宝在内不少同行的兴趣，并开始沟通将 CODE 开源的可能性。

作为一个面向内部设计的系统，CODE 的源代码中混杂了大量豆瓣专有系统的依赖，如果要开源出来需耗费不少的工作量，之前清风在 C2D2 的分享和段念在 QCon 上海 2013 大会的分享中都提到这一点：

CODE 在豆瓣内部是运行到 DAE 上面的，一些基础设施的维护都靠豆瓣平台组的工程师，如果要开源的话需要把豆瓣内部的依赖移除。另外豆瓣内部也有一套 Web 框架，CODE 用的就是这套内部框架，如果 CODE 开源，这套框架也需要一并开源。

虽然有种种麻烦，但豆瓣的工程师们仍然将大量工作之外的时间热情的投入到将 CODE 开源的工作当中。一开始，CODE 团队零星的将一些底层的基础库开源，包括 Python 版本的 git HTTP 实现 GPack，git SSH 实现 Maria，Pygit2 的 wrapper Ellen 等；2014 年 2 月 14 日，豆瓣 CODE 宣布将框架代码和周边项目代码全部公开！此时的 CODE 已经聚集了 85 位 committer 为其贡献代码，并且已经托管了 1916 个项目，其中大部分都是类似 CODE 这样的、由工程师自主发起的非官方项目。

技术说明

CODE 在豆瓣内部对应一个 DAE 应用，其实就是一个 Web 项目。CODE 目前最核心的工作是对 git 仓库的读写，另外就是 code review。

CODE 发展两年来一直在扩展自己的定位，从单纯的代码管理系统发展为可以将设计、产品都拉进来一同协作的平台，周边衍生出大量的附加项目，从 image diff 到类 Dropbox 的文件同步工具，可谓五花八门。所有这些功能都围绕一个核心：对 git 仓库的读写。

XTao：目前 CODE 支持 http 和 ssh 协议进行 clone 和 push 操作，http 目前我们使用的是 git_http_backend 这个开源项目，我们自己也用 Python 实现了一个 Grack（原项目基于 Ruby），即 GPack，目前还没有大规模的使用。ssh 我们用的是 CMGS 同学写的 Maria。

CODE 的 Web 版 git 操作的变化很大，一开始直接使用 subprocess 调用 shell 的 git 命令，然后解析命令返回的文本，python 对象化，然后供 Web 使用，后来因为性能问题开始使用 Pygit2，从此就是 git 命令和 Pygit2 混用，一直到现在。我们的这个库也开源了：Ellen。

Pygit2 是 libgit2 一个 Python 接口，在使用 Pygit2 和 libgit2 的过程中，也经常当小白鼠，也给 Pygit2 和 libgit2 提过一些 Pull Request。对 Pygit2 我们改动比较大，因为 CODE 使用模式问题，我们有一个自己的 fork 版本，跟上游的主要区别有：

把 libgit2 内嵌到 Pygit2 里面，做了静态编译，不再依赖系统的 libgit2 版本，方便升级 Pygit2。

一直采用最新版本的 libgit2，Pygit2 本身比较稳定，不能及时的使用 libgit2 最新的方法。

为了性能写了一些特殊的方法。

总体来说 libgit2 实现了核心的 git 接口，并不是很成熟，但是对于 Web 应用来说已经够用了。

以上的部分是与 git 操作紧密结合的部分。我们在 Web 界面也做了一些核心功能，主要包括 Pull Request 的实现，代码 diff 和 review 功能的实现，以及 Issue 和 Gist 系统。这里说一下代码 diff 和 review 功能，在实现的时候，一方面是用了 git 操作接口的 API，一方面是对于代码片段做了处理，在前端页面上支持 diff 的按行展开评论等一些功能。

同时，CODE 团队也在尝试分布式 git 方案，目前也已经有个原型了。

规则的建立

豆瓣曾经历过大桌子开会、强制大家做 review 的岁月，而 CODE 颠覆了这一情况。CODE 团队将 code review 视为 CODE 的第二个核心功能，认为促进工程师之间的沟通是 CODE 最大的成就之一。

CODE 为每个项目设置了三个角色，分为 owner（有全部权限）、committer（有 push 和 merge 权限）、member。review 机制根据项目的不同设置了不同的规则，如产品线级别的、需要对外发布的项目，基础库等项目都需要经过严格的 review，如东西团队对 review 设置了如下规则：

尊重他人，就事论事，对事不对人，毕竟每个人都写过烂代码；

PR 中的每一个 commit log 都应该可以和代码对应，方便 review；

尽量不要发太大的 PR，以免引起 reviewer 的恐慌；

建议保证一个 PR 的粒度和专注，最好不要出现一个 PR 里又有重构又加

新 feature 的情况，同样容易引起 reviewer 的恐慌；

提 PR 之前请确保在本地或测试环境一切正常；

reviewee 如果接受 reviewer 提出的修改意见，需要在修改提交以后知晓 reviewer，常见的做法可以是在 review comment 处回复（并带 commit 链接）；

评论中至少出现一个 lgtm 且保证 ci 通过之后 PR 才可以被合并；（注：lgtm 即 looks good to me 的缩写）

PR 合并者与提交者不能是同一个人；

PR 需从一个特定分支（分支的名字尽量能表达代码的功能）发往上游的 master 分支；

Model 的部分，如不紧急需要 unittest；

Web 的部分，如不紧急需要 webtest；

PR 合并后如引起 bug 或功能异常，并经查确为此 PR 引起，提交者需请全组攻城湿喝饮料或吃冰棍（由被请者决定）；

将 fork 的仓库与上游同步时，应使用 git fetch upstream && git rebase upstream/master（或 code sync -r ），而不是 git merge 或 code sync（这里 code 是指面向 CODE 系统的一个命令行工具），以保持清晰的提交历史，并防止覆盖他人的修改；

注意安全问题，对于 SQL 拼字符串，模版中有 |n 的，以及处理用户输入等地方都需要仔细 review，更多请参考 Web 安全开发指南

对于松散或娱乐性项目、小工具项目，并不会那么严格的 review，这也取决于 owner 自己，他可以借这个项目寻找到一位导师，来帮助他进行 review：

举一个具体的例子，例如东西团队的 Android 版本的开发，实际上最开始只是团队内部的一些成员想学习 Android 开发自发组织起来的，但一开始就找了移动组的同学来随时帮助 review。

对于 CODE 项目本身，所有工程师都可以向 CODE 上的任意项目提 PR，也都可以是 CODE 的 reviewer，同时所有工程师的代码都需要经过 review 才会被 merge 到 master 分支。

发展到现在，豆瓣的 review 基本上都是自发，很少遇到需要 review 的代码堆积的情况。代码讨论区里据说时不时会出现美女图，这可能是刺激工程师们

去 review 的因素之一；另外，CODE 系统本身也有奖励机制，鼓励大家去评论别人的代码。

CODE 系统的奖励机制主要有积分和勋章这两个部分。积分的规则主要就两个：

提交的 PR 被 merge，增加 100 点积分

提交的 PR 被评论，增加 5 点积分

目的就是鼓励多发 PR。一般来说，小 PR 要好过大 PR，不过有时候开发任务比较紧的时候，发出比较大的 PR 也是在所难免。

勋章系统在 CODE 早期阶段就做了进去，早期的奖励规则主要跟代码提交相关，例如给开源项目发过 Patch 并被 merge 会有相应的徽章。现在 CODE 团队对勋章系统有一些新的规划：

目前希望徽章系统可以独立出来，只是一套独立的 API，任何人任何产品线都可以去设置自己的奖励规则，让这种奖励变成不是一种公司行为，而是更小的行为。例如 antispam 组，可能就会有百人斩徽章，这个对于其他组可能就不是那么必要，当然如果你跨界帮助过 antispam 组，那么也有可能会获得这个徽章。

CODE 上没有设置惩罚机制。

测试

相比 Github，CODE 有一些非常实用的地方，比如在提交代码入库之前可以先在 CI 里面完成自动测试，reviewer 可以直接看到代码测试是通过（绿色）还是失败（红色）；代码完成 merge 之后还可以通过 DAE 直接往线上部署。持续集成、自动测试、监控、部署这些都是独立系统，与 CODE 都是靠 API 来进行交互。

对于测试的实现，豆瓣对开发者有明确的要求：

单元测试和集成测试的代码都是由开发者自己提供的，这是一种义务。测试的写法就是按照标准的 Python unittest 来写，断言比较推荐直接写 assert x == y。

投奔 git 之路

CODE 团队在下面的问答中分享了他们使用 git 的心得和经验，无论对正在使用 git 的团队还是还没有使用 git 的团队都是很好的参考。

InfoQ：你们当时从 SVN、Mercurial 转换到 CODE，这个过程好像还挺顺利

的？有没有遇到过一些阻力？如何化解的？

清风：其实过程并没有想像的那么愉快与顺利，这个过程 CODE 虚拟团队还是做出了很大努力的，日常的营销（拉人下水是很重要的），对于需求的快速满足，这些才是保障 CODE 逐渐被接受的重要因素。

开始的时候 CODE 功能比较简略，但大家毕竟都是用过 GitHub 的人，所以对功能上肯定是有要求的，经常会听到，某某某功能 GitHub 是有的，但 CODE 就没有，这个没法用啊，要不咱们还是尝试买 GitHub 企业版？这个时候其实也没有什么特别的方法，就是需要快速的去满足这个需求，并加以实现，可以比较自豪的说，CODE 的一些关键功能都是在 1-2 天，大一点的功能不超过 1 周实现的。

而且有趣的是，甚至有些功能在一开始的时候会认为和 GitHub 不一致，但后来发现习惯了甚至变成了一个设置项。最早 CODE 是双击某一行来进行 line comment 的，但 GitHub 是点击行首的“＋”号，进行 comment ，后来 CODE 也实现了行首点击“＋”号来 comment 的功能，但很多成员觉得双击很方便。这个还产生了很多的争论，有从方便性说的，有从 UI 人机交互规范谈起的，为了统一这两个派别，于是我们只好把这个做成一个设置项。

黄小毛：补充一下，豆瓣内部的团队是一个一个的迁移到使用 CODE 上的。在这个过程中，每个团队都会因为开发内容、使用工具、流程的不同，对 CODE 往往会提出不一样的需求，CODE 开发过程中都分别为这些团队做过相应的开发（其实也是因为这些功能之前都太简陋），一步一步的完善功能。常常是某个团队遇到了什么问题，CODE 开发人员都直接上门去解决，如果解决不到的，就加功能回来开发，一步一步得到使用团队的信任；同时因为 CODE 在豆瓣内部是架在 DAE 上的，常常需要 DAE 团队紧密的配合；各个产品部署上线的一套系统也逐渐使用 CODE 做为仓库之后，与 DAE 和 SA 的合作也逐步加深，就是这样慢慢的让整个产品线、平台部门、SA 部门，都转移到 CODE 上来的。

说个笑话，以前 CODE 开发还不稳定的时候，极少在白天上班时间上线，担心如果有什么状况，豆瓣几乎所有的工程师都要等着，耽误开发，可见豆瓣工程师对这个开发环境的依赖。这也是一步一步过来的。

InfoQ：对于 git，CODE 团队有没有特别的经验要分享？

CODE：大体上 git 就像大家那么用没有什么问题，只是开发过程中其实发现了一些 git 的问题，比如 git 编码存在问题，没有对 commit log 和 path 做 encoding ，在 Web 显示的时侯需要额外的处理。可以参考下这篇文章。

git 对大文件大仓库的支持不是很好，当一个项目变大的时候，适时的拆分一下。要不然就要像 Fackbook 那样去 hack hg 来支持大仓库。

GitHub 的 fork 模式导致磁盘空间被大量重复文件占用，GitHub 的工程师解决了这个问题，CODE 团队也在尽量解决这个重复文件的问题。

Java 实现的 git 库 CODE 团队还没有研究过，但是从 C、Python、Ruby 以及 Javascript 实现的 git 库并没有一个完美的解决方案，都需要混用 git 命令和第三方库。期待在不久的未来，libgit2 能完全取代 git 的核心库，成为一个完善的 git 库。目前 libgit2 还有一些内存泄漏的问题没有解决。

InfoQ：如果现在让你们重新设计 CODE，会在哪些方面做调整和重新的规划？是否会考虑基于类似 GitLab 进行二次开发？

CODE：现在的重构其实就是一次重新设计，最重要的方向就是把 API 和 UI 分离。

最开始做 CODE 的时候，其实考察过 GitLab，可惜当时的 GitLab 还不够成熟。如果当时的 GitLab 足够成熟，很有可能就会基于 GitLab 来开发了，不过，幸好没有发生这样的事情。

InfoQ：你是否会建议所有还在使用 svn 和 hg 的团队早日脱离苦海、投奔 git？

CODE：这个问题我一直是这么看的：git 固然强大，但是能否正确使用也是关键。我走访过很多团队，有些团队可能连版本控制工具本身都还没有掌握，例如每天提交一次，甚至一周提交一次，commit log 乱写，不使用分支，等等，这些都是使用者的问题。像 Facebook 最近就表示他们 hg 用的就不错（当然他们做了很多修改）。如果已经有了版本控制工具的正确使用习惯，那么我个人还是推荐 git。

原文链接：http://www.infoq.com/cn/articles/douban-code-2years

Android WebKit 消息处理

在 Android API 中，用 WebView 的形式向开发者提供 WebKit 的接口以及特性。WebView 其实是对 WebKit 的封装以及扩展，在 android4.4 里面，已经 WebKit 已经换成 Chromium(在后续博客中会对 android4.4 的 WebView/Chromium 进行详细讲解)。

WebKit 消息处理框架的搭建

整个 WebKit 主要分为 2 个线程，一个是 Ui 线程，也就是应用程序使用 WebView 所在的主线程，另一个 WebCore 线程。WebView 的消息处理，主要是 Ui 线程和 WebCore 线程的交互。一部分 Ui 线程向 WebCore 发送的命令操作消息，例如 LOAD_URL，另一部分是来自 Ui 的 touch 消息。

主要涉及到的类如图所示：

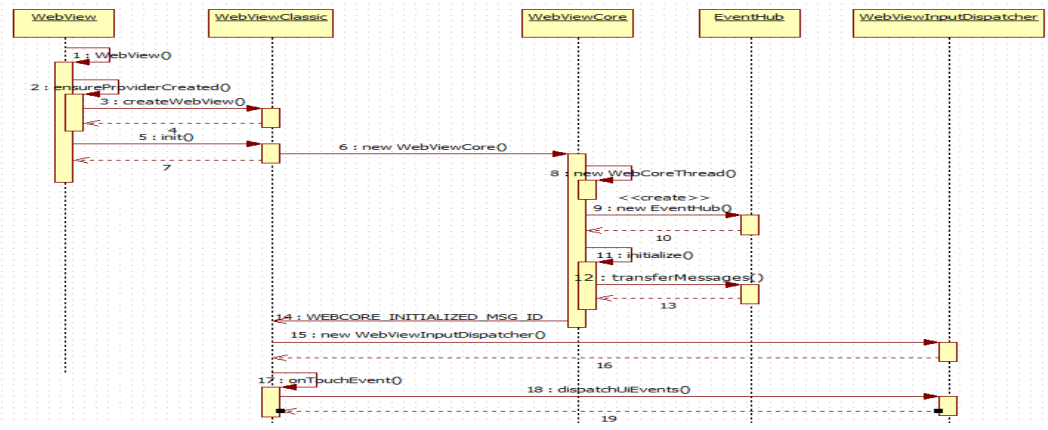


其中，WebViewClassic 是 WebView 的 Provider，或者说是 delegate，凡是涉及到消息处理或者需要跟 WebCore 交互的接口，都是直接调用 WebViewClassic 的同名函数。

WebViewInputDispatcher 就是用来处理 Ui 的 touch 事件的。后面会专门讲解 WebKit 的 touch 事件传递以及处理流程的。本章就不展开讲解了。

WebView 的初始化

时序图如下：

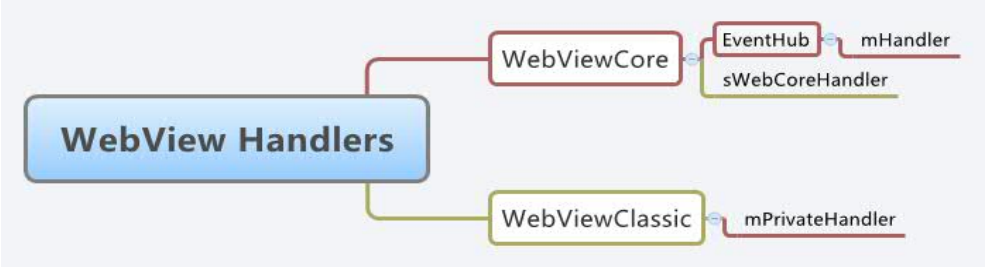


上图其实就是 WebView/WebKit 的初始化流程，主要是对 WebViewClassic, WebCore, WebViewInputDispatcher 进行初始化，为 WebKit 资源请求前做准备。WebView 构造函数中会调用 createWebView，这个过程其实就是创建 WebView 的 provider，实际返回的对象就是 WebViewClassic，紧接着 init 初始化 WebViewClassic，在 WebViewClassic 的 init 中会 new WebViewCore()，之后对 WebViewCore 进行初始化，在 WebViewCore 自己初始化完毕之后，表明现在 WebCore 已经可以处理来自 WebView 的消息了，包括 touch 事件，此时 WebViewCore 会发送消息给 WebViewClassic，也就是上图中看到的 WEBCORE_INITIALIZED_MSG_ID，WebViewClassic 收到此消息之后，就会初始化 touch 事件的分发器：WebViewInputDispatcher。这个流程结束之后，应用就可以通过 WebView，loadUrl 了。

从这个图中，我们看到，在 WebCore 的构造函数中会 new WebCoreThread，这个线程就是上面提到的 WebCore 线程。到此为止，我们已经可以看到有 2 个线程了，一个是 WebView 所在的 Ui 线程，另一个就是 WebCore 线程。

那么 WebKit 的消息处理究竟怎么工作的呢？

在 WebView 中，消息在线程间的分发使用的是 Handler。在 WebKit 的消息分发机制中的总共有三个 Handler 如下：



mPrivateHandler：

在 WebViewClassic 中创建，用来分发和处理 UI 相关消息，例如重绘，touch 事件，另外就是负责跟 WebCore 线程交互。

sWebCoreHandler：

在 WebCoreThread 中间，主要负责 WebViewCore 初始化、WebViewCoreWatchDog 心跳、WebCore 线程优先级别调整。

mHandler：

WebCore 线程消息循环最主要的 handler。任何需要调用 WebCore 接口的，

都需要通过 mHandler　send 到 WebCore 线程中去。

在 EventHub 的 transferMessages()中被 new 出来的，由于 transferMessages()实在 WebViweCore 的 initialize()中被调用的，所以，EventHub 的 mHandler 也是在 WebCore 线程中。

Ui 线程第一次向 WebCore 线程发送的消息，并没有直接被分发到 WebCore 线程中去。而是被缓存在 WebViewCore 中的 mMessages list 中，因为有可能在 WebKit 的消息处理框架还未初始化完毕，Ui 线程就已经开始向 WebCore 线程发送消息了。所以，当 WebViewCore 最后初始化完毕之后，会调用 transferMessages()，在 transferMessages 中将 mMessages 中的消息通过 mHandler 全部 send 到 WebCore 线程中去。

到这里已经应该很明白了：

Ui 线程的消息通过 WebViewClassic 的 mPrivateHandler 处理。

WebCore 线程的消息通过 EventHub 的 mHandler 和 WebViewCore 的 sWebCoreHandler 处理。各个 Handler 之间可以相互 send 消息到对方的消息队列中去。

原文链接：http://blog.csdn.net/feiyinzilgd/article/details/19019413

Tim Bray：2014 年软件之路

我们正处在构建软件的关键期。工具完善，服务端的开发者们很高兴，但说到客户端软件时，我们真不知去向何方。

前段欢乐时光。构建起的服务端代码技艺精湛，感谢你们。技术的扩展与提炼将继续持续下去。

一切都地能够与 HTTP 通信，而做到这点是很简单的。

一切都由 MVC 及类似的抽象层构建，并且有许多框架帮我们清晰稳健地完成这项工作。很多人还在使用 PHP 或者 Spring 构建重要的应用不得不说是件遗

憾的事情，虽然说这些新框架也没有强迫你使用它们。

我们仍在为选择动态类型和静态类型苦恼中。最后，妥协的理由却很好理解：两种语言之间都有好与坏。我两种都会使用，并且某些时候，使用的理由是显而易见的。请参见 Bánffy-Bray 准则。

并发

函数式编程渐渐在主流语言界享有一席之地。而原因在于关注性能就一定会涉及并发的问题。而一般情况下，人无法处理大量（或者根本不能处理）并发的，极易改变的共享事务。

许多人喜爱 Erlang，虽然它能很优雅地处理并发，甚至提供备用方案，但是它并不能大规模地用在生产中，因为它的数据类型和类概念与其他语言不同。

Clojure 的并发基础是函数式的，高效且优雅。而 Lisp 式的语法则是缺陷（从经验上来说，如果你不能像我一样理解 Lisp 的妙不可言时），而 Scala 虽然比 Java 简单，并且有像模像样的 Actor 模型，仍然十分繁杂。

NodeJS 本身不是函数式的，如果处理的一切都是事件，并且可以单线程的话，谁会在乎呢？但是我仍然在对 Node 的 JS 部分十分不满，待会儿再说明。

Go 给我的印象深刻，虽然它采用了 C、Java、Ruby、Clojure 等语言的做法并不能使我开心一笑。我感觉它的类型系统提供了许多针对对象的实用工具，我强烈感到 Goroutines 和类型管道是非常出色的设计，开发者可以够顺利地写出函数式代码。这种做法容易，直接又可读性好，我考虑下一个重大项目的服务端代码使用 Go 语言编写。

如果上面的这些都不符合要求，我们还考虑使用这些由高手打造的 Rust、Elixir、Dart 等语言。

存储

现在各种持久化方案十分成熟。我已经很长时间不再在性能关键的运行时系统中使用关系型存储了；同时它仍有用武之地并有许多开源的选择。

这些关系型数据库之后出现的方案也足够完善。从轻量级的内存缓存到可以操作巨型数据的软件，都有对应的软件可供挑选。你可以看看 Cassandra，如果你最近听过 Adrian Cockcroft 的演讲，知道 Netfilx 如何使用它的时候，你就会感到吃惊。

高手们都把磁盘当成新式磁带一样，找到合理地使用它的方式。

而另一方面……

客户端的混乱

情况十分糟糕。你需要造三遍轮子：Web、iOS、Android。我们缺乏人才，而这样的开发环境十分浪费，一直折磨着我们。

移动端太糟

此处略去 Android 和 iOS 的具体差异，在工程上来说，这些差异不是十分显著，但是，仍然有以下糟糕之处：

首先，你需要开发两种不同的客户端。

更新周期十分缓慢，以基于浏览器的 App 比较，Android 上花的几个小时，放在 iOS 上就需要几天时间。更糟糕的是你并不能指望移动用户接收你的每次更新。发现了一个导致数据丢失，违反用户协议隐私条款的 bug？足够让你吃尽苦头了。

设备非常吃内存、CPU，耗电量猛增。

表单的加载越来越慢，出现进度条需要等待。

你没有编程语言的选择权，如果你厌恶 ObjC 和 Java 的话，就需要考虑换工作了。

单元测试很操蛋。

有利于用户但不利于开发者的你而言，移动端对于 UX（用户体验）的要求很高，没有捷径可寻，同时需要灵感涌现和反复尝试。

使用互联网的正确方式是点击浏览器上方的搜索栏，输入你感兴趣的内容，点击搜索，点击结果链接，就可以得到你想要的信息。但是无论你在移动设备上搜索什么，你都需要安装相应的应用，同时意味着在手机应用商店还有另外一层搜索，而搜索结果比不上 Google 或者 Bing 的质量。

你不能赚钱。严肃点来说，苹果总是谈论到他们在应用商店外花的成千上万的钱。我还没有听说过谁靠着应用商店正正经经地赚了许多的钱。

当然，HTML5 热潮正当其时，告诉人们，如果人们开发的是移动 Web 应用的时候，所有的不利之处（尤其是第一条）就将消解。

但是……

浏览器同样很糟糕 虽然这是个老生常谈的话题，但是还是看不出为什么它如此充满争议。

JavaScript 不可理喻之处：

```
1       > [5, 10, 1].sort();
2       [ 1, 10, 5 ]
```

以上的例子还有很多。所以就有了 CoffeeScript 和 Dart 这类语言。他们都在想办法解决这些刻意回避的问题。

浏览器的 API 也很糟糕，所以人们都基于 jQuery（以及类似的库）看作在此之上编程的底层库，因此让 JS 变成了 Web 时代的汇编语言。

于是，在实际构造应用程序的时候，你就需要挑选更高层次的框架。网上有很多这样的框架，很容易就能搜索到相关的信息，像这个：Rich JavaScript Applications – the Seven Frameworks (Throne of JS, 2012)。但是这个已经是 18 个月前的信息了，放到现在可能完全是错误的。你可能会喜欢有更多选择，但是这样下去会造成“寒武纪大爆发”式的增长。我觉得 2113 年的软件架构师会喜欢研究这些问题的。

（同时，请阅读：Tero Piirainen 的 Frameworkless JavaScript）

CSS 也很糟糕。我本想解释这一点，不过已经有这篇文章：Why Sass?，所以我不必这么做了。同时请查看：Less vs Sass vs Stylus，看看有没有我提到的“寒武纪大爆发”问题？

现在还没有可以像应用商店一样能筛选应用程序大小的地方。

好了好了，我知道每个以 Web 为中心的大型会议，那些眼睛闪耀光芒的，充满激情，真心相信浏览器的信徒们会向你展示 HTML5 的酷炫之处。而且他们也可以使用加速度传感器配合麦克风写出移动设备上的独特 APP 呢。

那，为什么没有那么多人这么做呢？提示：请看上面列出的几条观点。

我在说”移动端太糟“，不是表面工程软件的糟糕；而事实上，Cocoa Touch 和 Android app framework 都在 GUI 构建方面做得很好，吸取了很多历史教训。关键是，你所想要放到 UI 上的东西，都会有一个简单的，符合标准并经过测试的方法，一般会成为 Google 和 Stack Overflow 网站上的第一条内容。

但是看看投入到 Web 技术的所有精力吧，它真的能跟上当今移动端的技术

进步吗？也许吧，那或许是在 Google 和苹果的精英团队及世界上顶尖的 GUI 工程师对它进行一番筛选扩充以后的事情。所以，我有点期待稳扎稳打，一往无前的时刻了。

收益减少

我是个老古董，仍然记得第一波 Web 应用兴起的时候，横扫那些用 Visual Basic、 Motif、Java、Win32 编写的一整代软件，正是因为人们喜欢用浏览器处理所有的事务。

当然，15 分钟后，软件的 VIP 用户们就开始诉说浏览器界面过于笨重，反应不够灵活，而我们得找到 B 方案，我发现那些 VIP 客户们都接受了私有的 B 方案。于是现在我们有了 B 方案，至少它符合标准。

但是，我仍然半信半疑。是的，我喜欢让应用良好地相应手势，物件有滑进淡出效果，但是那也只是锦上添花，离完美，也就是 80/20 法则所说的那样还很远——放在服务器上的良好设计的 Web 应用正常运行，并保持良好的投资回报率。我非常讨厌屏幕上四个独立滚动的，用 JS 控制滚动，看上去外观非常拙劣的区域。我稍后会写一些出色的单页应用，故意来一些缩进让它看上去有点偏。我尤其讨厌让非技术伙伴，或者亲友们遇到上面的糟糕情况，而我得花时间解释原委。

接下来？

服务端并无惊喜，诸事顺利，一切如往日美好。

而客户端，我什么也不知道。由历史原因造成纷繁复杂的做法最终会被那些简单的，满足 80/20 法则的做法所替代。如果这正是未来的方向的话，应该不是来自我们这个方向，显然现在仍然让我们困惑不已。或许我们还得长期应付这种一个客户端做三份的情况。

原文链接：http://blog.jobbole.com/58671/

MongoDB 与内存

但凡初次接触 MongoDB 的人，无不惊讶于它对内存的贪得无厌，至于个中缘由，我先讲讲 Linux 是如何管理内存的，再说说 MongoDB 是如何使用内存的，答案自然就清楚了。

据说带着问题学习更有效，那就先看一个 MongoDB 服务器的 top 命令结果：

```
shell> top -p $(pidof mongod)
Mem:  32872124k total, 30065320k used,  2806804k free,   245020k buffers
Swap:  2097144k total,      100k used,  2097044k free, 26482048k cached
 VIRT  RES  SHR %MEM
1892g  21g  21g 69.6
```

这台 MongoDB 服务器有没有性能问题？大家可以一边思考一边继续阅读。

先讲讲 Linux 是如何管理内存的

在 Linux 里(别的系统也差不多)，内存有物理内存和虚拟内存之说，物理内存是什么自然无需解释，虚拟内存实际是物理内存的抽象，多数情况下，出于方便性的考虑，程序访问的都是虚拟内存地址，然后操作系统会把它翻译成物理内存地址。

很多人会把虚拟内存和 Swap 混为一谈，实际上 Swap 只是虚拟内存引申出的一种技术而已：操作系统一旦物理内存不足，为了腾出内存空间存放新内容，就会把当前物理内存中的内容放到交换分区里，稍后用到的时候再取回来，需要注意的是，Swap 的使用可能会带来性能问题，偶尔为之无需紧张，糟糕的是物理内存和交换分区频繁的发生数据交换，这被称之为 Swap 颠簸，一旦发生这种情况，先要明确是什么原因造成的，如果是内存不足就好办了，加内存就可以解决，不过有的时候即使内存充足也可能会出现这种问题，比如 MySQL 就有可能出现这样的情况，解决方法是限制使用 Swap：

```
shell> sysctl -w vm.swappiness=0
```

查看内存情况最常用的是 free 命令：

shell> free -m

	total	used	free	shared	buffers	cached
Mem:	32101	29377	2723	0	239	25880
-/+ buffers/cache:		3258	28842			
Swap:	2047	0	2047			

新手看到 used 一栏数值偏大，free 一栏数值偏小，往往会认为内存要用光了。其实并非如此，之所以这样是因为每当我们操作文件的时候，Linux 都会尽可能的把文件缓存到内存里，这样下次访问的时候，就可以直接从内存中取结果，所以 cached 一栏的数值非常的大，不过不用担心，这部分内存是可回收的，操作系统会按照 LRU 算法淘汰冷数据。除了 cached，还有一个 buffers，它和 cached 类似，也是可回收的，不过它的侧重点在于缓解不同设备的操作速度不一致造成的阻塞，这里就不多做解释了。

知道了原理，我们就可以推算出系统可用的内存是 free + buffers + cached：

```
shell> echo "2723 + 239 + 25880" | bc -l
28842
```

至于系统实际使用的内存是 used - buffers - cached：

```
shell> echo "29377 - 239 - 25880" | bc -l
3258
```

除了 free 命令，还可以使用 sar 命令：

```
shell> sar -r
kbmemfree kbmemused    %memused kbbuffers    kbcached
   3224392    29647732        90.19       246116    26070160
   3116324    29755800        90.52       245992    26157372
   2959520    29912604        91.00       245556    26316396
   2792248    30079876        91.51       245680    26485672
   2718260    30153864        91.73       245684    26563540
shell> sar -W
```

pswpin/s pswpout/s

0.00 0.00

0.00 0.00

0.00 0.00

0.00 0.00

0.00 0.00

希望你没有被%memused 吓到，如果不幸言中，请参考 free 命令的解释。

再说说 MongoDB 是如何使用内存的

目前，MongoDB 使用的是内存映射存储引擎，它会把磁盘 IO 操作转换成内存操作，如果是读操作，内存中的数据起到缓存的作用，如果是写操作，内存还可以把随机的写操作转换成顺序的写操作，总之可以大幅度提升性能。MongoDB 并不干涉内存管理工作，而是把这些工作留给操作系统的虚拟缓存管理器去处理，这样的好处是简化的 MongoDB 的工作，但坏处是你没有方法很方便的控制 MongoDB 占多大内存，事实上 MongoDB 会占用所有能用的内存，所以最好不要把别的服务和 MongoDB 放一起。

时候，即便 MongoDB 使用的是 64 位操作系统，也可能会遭遇臭名昭著的 OOM 问题，出现这种情况，多半是因为限制了虚拟内存的大小所致，可以这样查看当前值：

```
shell> ulimit -a | grep \'virtual\'
```

多数操作系统缺省都是把它设置成 unlimited 的，如果你的操作系统不是，可以这样修改：

```
shell> ulimit -v unlimited
```

不过要注意的是，ulimit 的使用是有上下文的，最好放在 MongoDB 的启动脚本里。

有时候，出于某些原因，你可能想释放掉 MongoDB 占用的内存，不过前面说了，内存管理工作是由虚拟内存管理器控制的，所以通常你只能通过重启服务来释放内存，这听起来有点恶心，不过幸好可以使用 MongoDB 内置的 closeAllDatabases 命令达到目的：

```
mongo> use admin
mongo> db.runCommand({closeAllDatabases:1})
```

另外，通过调整内核参数 drop_caches 也可以释放缓存：

```
shell> sysctl -w vm.drop_caches=1
```

平时可以通过 mongo 命令行来监控 MongoDB 的内存使用情况，如下所示：

```
mongo> db.serverStatus().mem:
{
        "bits" : 64,
        "resident" : 22346,
        "virtual" : 1938524,
        "supported" : true,
        "mapped" : 962283
}
```

还可以通过 mongostat 命令来监控 MongoDB 的内存使用情况，如下所示：

```
shell> mongostat
mapped    vsize       res faults
   940g    1893g    21.9g          0
   940g    1893g    21.9g          0
   940g    1893g    21.9g          0
   940g    1893g    21.9g          0
   940g    1893g    21.9g          0
```

其中内存相关字段的含义是：

mapped：映射到内存的数据大小

visze：占用的虚拟内存大小

res：实际使用的内存大小

注：如果操作不能再内存中完成，结果 faults 列的数值不会是 0，视大小可能有性能问题。

在上面的结果中，vsize 是 mapped 的两倍，而 mapped 等于数据文件的大小，所以说 vsize 是数据文件的两倍，之所以会这样，是因为本例中，MongoDB 开启

了 journal，需要在内存里多映射一次数据文件，如果关闭 journal，则 vsize 和 mapped 大致相当。

如果想验证这一点，可以在开启或关闭 journal 后，通过 pmap 命令来观察文件映射情况：

```
shell> pmap $(pidof mongod)
```

到底 MongoDB 配备多大内存合适？宽泛点来说，多多益善，如果要确切点来说，这实际取决于你的数据及索引的大小，内存如果能够装下全部数据加索引是最佳情况，不过很多时候，数据都会比内存大，比如本文说涉及的 MongoDB 实例：

```
mongo> db.stats()
{
        "dataSize" : 1004862191980,
        "indexSize" : 1335929664,
}
```

本例中索引只有 1G 多，内存完全能装下，而数据文件则达到了 1T，估计很难找到这么大内存，此时保证内存能装下热数据即可，至于热数据有多少，这就是个比例问题了，取决于具体的应用。如此一来内存大小就明确了：内存 > 索引 + 热数据。

原文链接:http://blogread.cn/it/article/4185

《淘宝技术这十年》读书笔记



这本书很有趣，故事叙述好玩，且价值颇高；

它让更多人了解并熟悉淘宝，让我们近距离观察大公司成长过程中遇到的问题；

我将从三个方面来做这个概要版的读书笔记；

淘宝的升级路线叙述了淘宝的成长路线，如果能引起大家的兴趣，那最好，书中有详实有味的数据供人品味；

淘宝创新的技术一节总结淘宝在壮大的道路中所创造的技术，其实很多在我们的日常应用中都能找到影子，所不同的是能够支撑的规模；

最后一节出自这本书的最后一章，牛 P 列传；本书的作者采访了淘宝中的牛人，道出他们的成长经历，同时也能看到这些高人对新人的建议；

豆瓣评分 7.4 分，我给 4 星；

淘宝的升级路线

2003 年 4 月 7 日，马云，在杭州，成立了一个神秘的组织。

为了在最短的时间做出一个网站来，为了和时间赛跑，淘宝的第一版源代码直接

从国外买来，架构为 LAMP(Linux+Apache+MySQL+PHP)；买来之后做了些本地化的修改，对数据库进行读写分离，一个月之后上线，兵贵神速。

最初上线的淘宝只有一台服务器，随着访问量的增加，将数据库服务器独立出来，同时，增加一台 Email 服务器；商品检索功能最初的实现就是使用 sql 中的 like 搜索，对数据库资源消耗严重，最快的改进招数就是找老爹要，使用阿里巴巴的 iSearch 模块替换 like 方式；

在 2003 年底，MySQL 已经撑不住了，技术的替代方案非常简单，就是换成 Oracle。换 Oracle 的原因除了它容量大、稳定、安全、性能高之外，还有人才方面的原因。在 2003 年的时候，阿里巴巴已经有一支很强大的 DBA 团队。其中，ACE 人才就有 4 名；

到 2004 年上半年，淘宝决定更换开发语言，将 PHP 更改为 JAVA；

到 07 年，淘宝已经用上了 IBM 的小型机、Oracle 的数据库、EMC 的存储，这些东西都是很贵的，那些年可以说是花钱如流水。有人说过“钱能解决的问题，就不是问题”，但随着淘宝网的发展，不久以后，钱已经解决不了问题。

07 年之后，淘宝就开始了真正的壮大之路，走上了创造技术的大道；

07 年之前，图片存储使用的是 NetAPP 的文件存储系统，但未对小文件存储进行有针对的优化 ，且扩容成本高昂，网络连接数经常达到极限；在这种背景下，淘宝研发了 TFS（淘宝文件系统）；2007 年 6 月，TFS 正式上线运营。在生产环境中应用的集群规模达到了 200 台 PC Server(146G*6 SAS 15K Raid5)，文件数量达到上亿级别；系统部署存储容量：140TB；实际使用存储容量： 50TB；单台支持随机 IOPS200+，流量 3MBps。

淘宝创新的技术

简单总结下淘宝创造了哪些技术：

TFS

淘宝文件系统，上述已说明；

Tair：淘宝 KV 缓存系统

04 年多隆基于 BerkeleyDB 开发了 TBstore，一个 key-value 的缓存系统；

07 年，多隆参考 memcache 内存结构，为 UIC 开发了 TDBM，另一个缓存系统；之后合并这两个缓存，并在此基础上进一步扩展，开发出 Tair，底层支持 5 种存

储引擎（BerkeleyDB、Redis，TT、fDB、memcache）

目前，Tair 已开源；

HSF：高性能服务框架

一个分布式 PRC 框架，用于系统间调用

service 定于基于 OSGI 方式，通讯层采用 TCP/IP 协议

Notify：消息中间件

分布式消息中间件，用于替代 MQ

支持消息订阅、发布和消费

TDDL：淘宝分布式数据访问层

数据访问路由，使得数据访问对应用层的开发透明，一个表的数据可以分布在多台结点上，而应用的编码同操作单节点 DB 一样；

支持一次写入，多点读取

数据存储的自由扩展， 不再受限单机容量和速度瓶颈

TbSession 框架

Session 客户端存储，session 存到 cookie 中

实现分布式环境下的 session 服务器端存储

session 配置统一管理

支持动态更新

从牛人身上看到的

（以下均是花名，淘宝特色）

正明

博士期间开发了开源 LVS，2000~2006 年在国防科大教书；多次创业，做出来的产品可圈可点，但终究无法得到市场的认可；能做出好的产品并一定就能成功，市场推广是门大学问；

08 年进淘宝，做 LVS+HAproxy 的普及，现在负责 CDN 系统的改良工作 ；

纯粹的技术人才，如果找不到互补的伙伴，加入一个大公司，也不失为一个好的选择；

毕玄

06 年加入淘宝，负责淘宝的高性能服务框架，著作《OSGi 原理与最佳实践》，

推广 OSGi 在中国的应用；

看看他的每日时间分配：代码 3 小时、剩下来很多时间看技术文章或图书，每隔一段时间跳出来想想将来干什么好；

作为淘宝晋升委员会的评委，他给同事们的建议是：

跳出你的位置，想想公司到底面临什么问题，用什么方法来解决？

晋升是看你对公司有哪些贡献，而不是看你的技术有多么牛；

放翁

淘宝开放平台项目负责人

职业经历

在国企东方通做了 4 年，之后转入阿里巴巴；

放翁的入职经历值得新人学习；在入职阿里巴巴后的初期，他干的是琐事，负责开发 web 页面之类的活；期间主动学习不归他负责的 WEB 框架，并多次修改框架 bug，最终被上层发现并赏识，从而风生水起；

他新人的建议：

做事要先思考再问人，不断打破自己的一些想法 ；

任何一个公司，都无法做到绝对公平，最终只会有一小部分人得到机会； 抱怨无用，只有不断努力，争取机会；

多隆

淘宝创始人之一

多隆沉浸在程序世界，不受干扰；

公司郊游回来的火车上，他在写代码

给工程师的建议

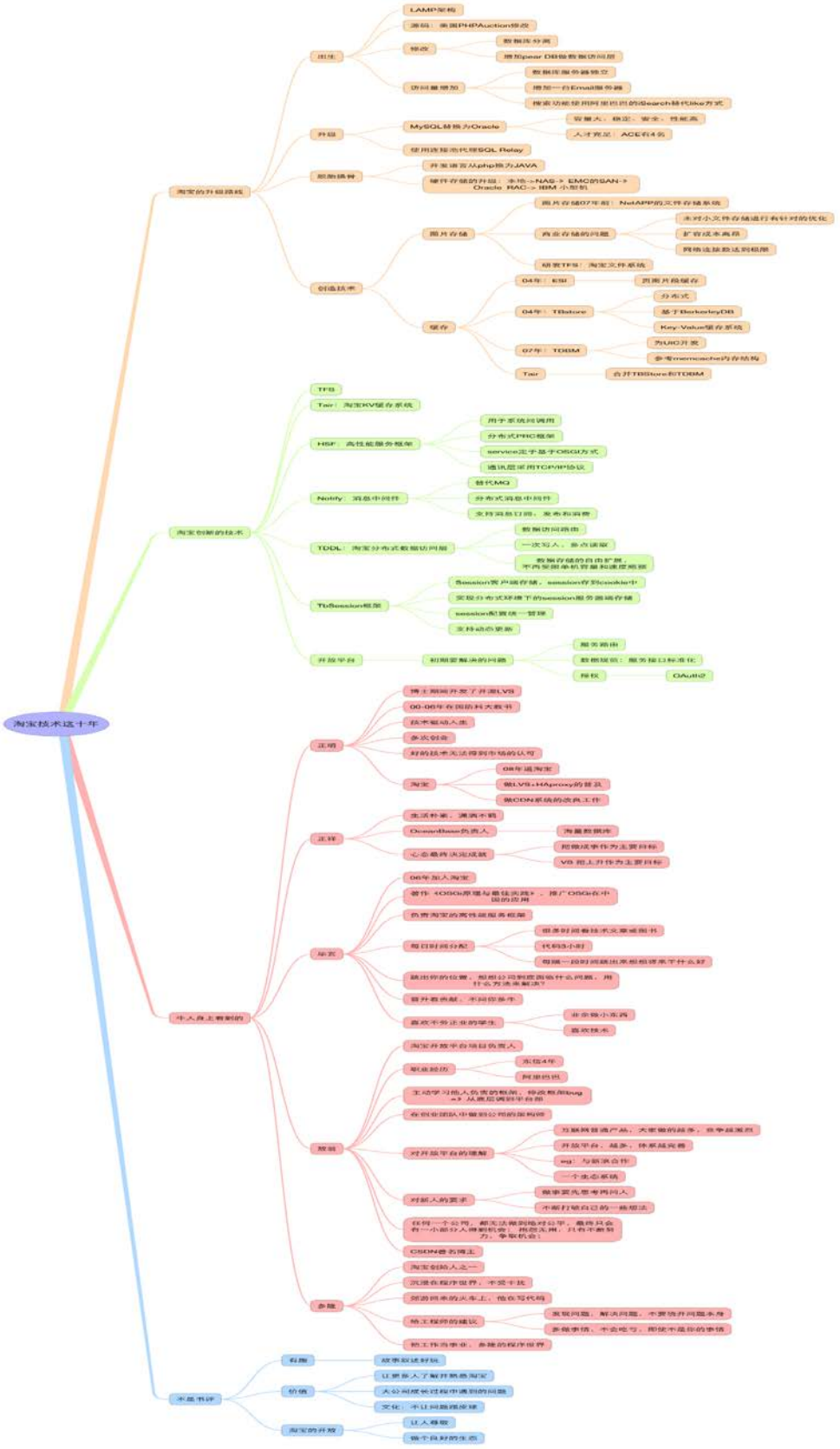
发现问题，解决问题，不要绕开问题本身

多做事情，不会吃亏，即使不是你的事情

把工作当事业，多隆的程序世界

亚马逊地址: 《淘宝技术这十年》

附上思维导图笔记(点击放大)：



原文链接：http://www.cnblogs.com/me115/p/3545054.html

探索 Hibernate 新 TableGenerator 机制

Hibernate 从 3.2.3 之后引入了两个新的主键生成器 TableGenerator 和 SequenceStyleGenerator。为了保持与旧版本的兼容，这两个新主键生成器在默认情况下不会被启用，而不启用新 TableGenerator 的 Hibernate 在提供 JPA 的 @TableGenerator 注解时会有 Bug。本文从这个 Bug 开始，讲述了如何启用新 TableGenerator，并演示了新 TableGenerator 的完整用法。

从 initialValue 说起

问题的发现源自对 JPA 中 TableGenerator 的测试。测试的环境有这样几个条件：

为方便查询的测试，Employee 表格在初始化时会导入部分记录，这部分记录的主键在初始脚本中手动写好，比如 1、2、3、4。（参看文章所附示例代码中的 import_data.sql 文件）。

Employee 实体使用 TableGenerator 主键生成器，initialValue 的值设置为 10。

在单元测试中添加新的 Employee 记录。

Employee 实体类的代码参看清单 1：

清单 1. Employee 实体类

```
1  @Entity @Table(name="emp3")
2  public class Employee3 {
3
4  @TableGenerator(name="id_gen",table="id_gen",initialValue=10) @Id
5      @GeneratedValue(strategy=TABLE,generator="id_gen")
6      private long id;
7      private String firstName;
8      private String lastName;
9      .....
   }
```

@TableGenerator 的配置参数 initialValue 指的是主键生成列的初始值，这

在 @TableGenerator 的 API 文档中写得很清楚。现在 initialValue 值设置为 10，那么在单元测试中用 JPA 添加新的 Employee 记录时，新记录的主键会从 11 开始，不会与已有的数据发生冲突（参看文章所附示例代码中 src/java/test/sample/case3/OldInitialValue.java）。执行的结果出乎意料，测试报错，说是主键重复。错误信息如清单 2 所示：

清单 2. 主键重复错误信息

```
1 11:23:40,220 ERROR SqlExceptionHelper:144 - Duplicate entry '1' for key 'PRIMARY'
```

这实在令人困惑。如果 initialValue 的含义不是初始值，那还能是什么呢？问题其实出在程序所用的 JPA 提供者（Hibernate）上面。如果改用其他 JPA 提供者，估计不会出现上面的问题（未验证）。Hibernate 之所以会出现这种情况，并非无知，也不是不尊重标准，而有它自身的原因，这在文章后面会提到。现在，为了把问题讲清楚， 有必要先谈谈 JPA 主键生成器选型的问题，了解一下 @TableGenerator 在 JPA 中的特殊地位。

JPA 主键生成器选型

JPA 提供了四种主键生成器，参看表 1：

表 1. JPA 的四种主键生成器

生成器名称	描述
AUTO	由 JPA 提供者根据数据库自行决定生成算法。
IDENTITY	由数据库的自增列提供主键值。
SEQUENCE	由数据库 Sequence 对象提供主键值。

TABLE	由 JPA 提供者通过创建数据库表来记录生成的主键值。

一般来说，支持 IDENTITY 的数据库，如 MySQL、SQL Server、DB2 等，AUTO 的效果与 IDENTITY 相同。IDENTITY 主键生成器最大的特点是：在表中插入记录以后主键才会生成。这意味着，实体对象只有在保存到数据库以后，才能得到主键值。用 EntityManager 的 persist 方法来保存实体时必须在数据库中插入纪录，这种主键生成机制大大限制了 JPA 提供者优化性能的可能性。在 Hibernate 中通过设置 FlushMode 为 MANUAL，可以将记录的插入延迟到长事务提交时再执行，从而减少对数据库的访问频率。实施这种系统性能提升方案的前提就是不能使用 IDENTITY 主键生成器。

SEQUENCE 主键生成器主要用在 PostgreSQL、Oracle 等自带 Sequence 对象的数据库管理系统中，它每次从数据库 Sequence 对象中取出一段数值分配给新生成的实体对象，实体对象在写入数据库之前就会分配到相应的主键。

上面的分析中，我们把现实世界中的关系数据库分成了两大类：一是支持 IDENTITY 的数据库，二是支持 SEQUENCE 的数据库。对支持 IDENTITY 的数据库来说，使用 JPA 时变得有点麻烦：出于性能考虑，它们在选用主键生成策略时应当避免使用 IDENTITY 和 AUTO，同时，他们不支持 SEQUENCE。看起来，四个主键生成器里面排除了三个，剩下唯一的选择就是 TABLE。由此可见，TABLE 主键生成机制在 JPA 中地位特殊。它是在不影响性能情况下，通用性最强的 JPA 主键生成器。

TableGenerator 有新旧之分？

JPA 的 @TableGenerator 只是通用的注解，具体的功能要由 JPA 提供者来实现。Hibernate 中实现该注解的类有两个，一是原有的 TableGenerator，类名为 org.hibernate.id.TableGenerator，这是默认的 TableGenerator。二是新 TableGenerator，指的是 org.hibernate.id.enhanced.TableGenerator。当用 Hibernate 来提供 JPA 时，需要通过配置参数指定使用何种 TableGenerator 来提供相应功能。

在 4.1 版本的 Hibernate Reference Manual 关于配置参数的章节中（网址可

从参考资源中找到）可以找到如下说明：

我们建议所有使用 @GeneratedValue 的新工程都配置 hibernate.id.new_generator_mappings=true 。因为新的生成器更加高效，也更符合 JPA2 的规范。不过，要是已经使用了 table 或 sequence 生成器，新生成器与之不相兼容。

还可以再参考一下 HHH-4884 和 HHH-4690 ，里面有 Hibernate 开发人员对这些问题的看法。

综合这些资源，可以得到如下结论：

如果不配置 hibernate.id.new_generator_mappings=true，使用 Hibernate 来提供 TableGenerator 时，JPA 中 @TableGenerator 注解的 initialValue 参数是无效的。

Hibernate 开发人员原本希望用新 TableGenerator 替换掉原有的 TableGenerator，但这么做会导致已经使用旧 TableGenerator 的 Hibernate 工程在升级 Hibernate 后，新生成的主键值可能会与原有的主键冲突，导致不可预料的结果。为保持兼容，Hibernate 默认情况下使用旧 TableGenerator 机制。

没有历史负担的新 Hibernate 工程都应该使用 hibernate.id.new_generator_mappings=true 配置选项。

现在回到清单 1 所示的问题，要解决这个问题只需在 persistence.xml 文件中添加如下一行配置即可：

清单 3. 添加新的配置行

```
1 &lt;property name="hibernate.id.new_generator_mappings"
value="true"/&gt;
```

使用新 TableGenerator 后就可以放心地在 JPA 中使用 initialValue 参数了，不过，这只是新 TableGenerator 的一个好处，我们接下来还可以看看新 TableGenerator 带来的更多用法。

新 TableGenerator 的更多用法

新 TableGenerator 除了实现 JPA TableGenerator 注解的全部功能外，还有其他 JPA 注解没有包含的功能，其配置参数共有 8 项。新 TableGenerator 的 API 文档详细解释了这 8 项参数的含义，但很奇怪的是，Hibernate API 文档中

给出的是 Java 常量的名字，在实际使用时还需要通过这些常量名找到对应的字符串，非常不方便。用对应字符串替换常量后，可以得到下面的配置参数表：

表 2. 新 TableGenerator 配置参数表

序号	参数名	默认值	含义
1	table_name	hibernate_sequence	辅助表的表名
2	value_column_name	next_val	存放序列值的列名
3	segment_column_name	sequence_name	存放序列名的列名
4	segment_value	default	序列名

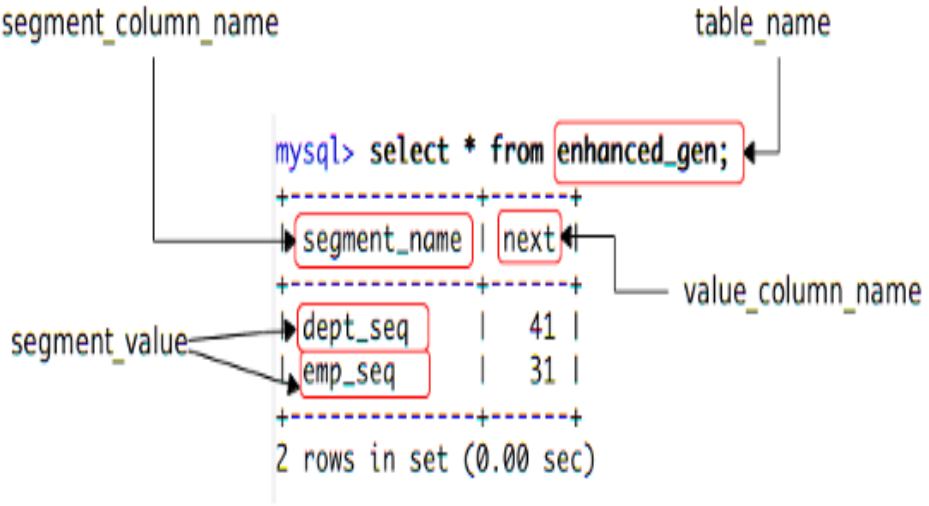
5	segment_value_length	255	序列名所在列数据类型长度
6	initial_value	1	指定序列的初始值
7	increment_size	1	指定序列默认的递增量
8	optimizer	依 increment_size 的取值而定	指定序列的优化器

在描述各个参数的含义时，表中多次提到了“序列”，在这个表里的意思相当于 sequence，也相当于 segment。这里反映出术语的混乱，如果在 Hibernate 文档中把两个英文单词统一起来，阅读的时候会更加清楚。新 TableGenerator 的 8 个参数可分为两组，前 5 个参数描述的是辅助表的结构，后 3 个参数用于配置主键生成算法。

先来看前 5 个参数，下图是本文示例程序用于主键生成的辅助表，把图中

的元素和新 TableGenerator 前 4 个配置参数一一对应起来，它们的含义一目了然。

图 1. 辅助表



第 5 个参数 segment_value_length 是用来确定序列名称所在列的长度，即序列名所能使用的最大字符数。从这 5 个参数的含义可以看出，新 TableGenerator 支持在同一个表中放下多个主键生成器，从而避免数据库中为生成主键而创建大量的辅助表。

后面 3 个参数用于描述主键生成算法。第 6 个参数指定初始值。第 7 个参数 increment_size 确定了步长。最关键的是第 8 个参数 optimizer。optimizer 的默认值一栏写的是“依 increment_size 的取值而定”，到底如何确定呢？

为搞清楚这个问题，需要先来了解一下 Hibernate 自带的 Optimizer。

Hibernate 自带的 Optimizer

Optimizer 可以翻译成优化器，使用优化器是为了避免每次生成主键时都会访问数据库。从 Hibernate 官方文档中找不到优化器的说明，需要查阅源码，在 org.hibernate.id.enhanced.OptimizerFactory 类中可以找到这些优化器的名字及对应的实现类，其中优化器的名字就是新 TableGenerator 中 optimizer 参数中能够使用的值：

表 3. Optimizer 名字及实现类

表 3. Optimizer 名字及实现类

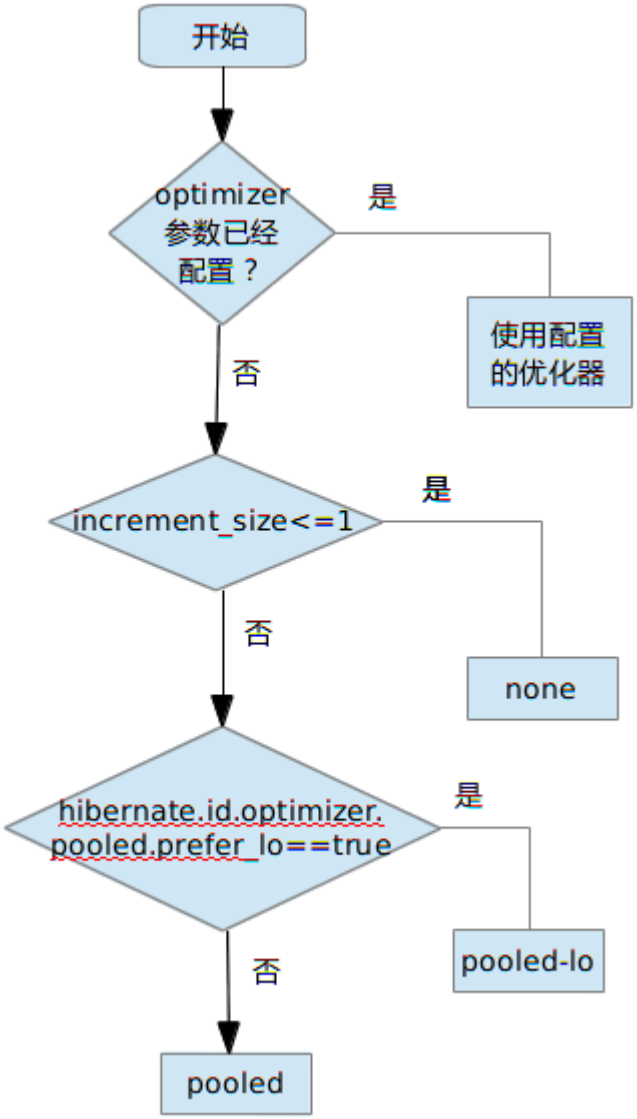
Optimizer 名字	类名	特点
none	NoopOptimizer.class	没有任何优化，每次主键生成都需要访问数据库。
hilo	HiLoOptimizer.class	hilo 算法实现的主键生成机制，数据库中的值是 bucket 的序号。
legacy-hilo	LegacyHiLoAlgorithmOptimizer.class	旧 hilo 算法。

pooled	PooledOptimizer.class	也使用 hilo 算法，不同之处在于 bucket 内部数值保存在数据库中。
pooled-lo	PooledLoOptimizer.class	算法与 pooled 完全相同，但保存在数据库中的值不同于 pooled。

Hibernate 自带了 5 种优化器，那么现在就可以加到上一节提到的问题了：默认情况下，新 TableGenerator 会选择哪个优化器呢？

又一次，在 Hibernate 文档中找不到答案，还是要去查阅源码。通过分析 TableGenerator，可以看到 optimizer 的选择策略。具体过程可用下图来描述：

图 2. 选定优化器的过程



可以看出，hilo 和 legacy-hilo 两种优化器，除非指定，一般不会在实践中出现。接下来很重要的一步就是判断 increment_size 的值，如果 increment_size 不做指定，使用默认的 1，那么最终选择的优化器会是“none”。选中了“none”也就意味着没有任何优化，每次主键的生成都需要访问数据库。这种情况下 TableGenerator 的优势丧失殆尽，如果再用同一张表生成多个实体的主键，构造出来的系统在性能上会是程序员的噩梦。

在 increment_size 值大于 1 的情况下，只有 pooled 和 pooled-lo 两种优化器可供选择，选择条件由布尔型参数 hibernate.id.optimizer.pooled.prefer_lo 确定，该参数默认为 false，这也意味着，大多数情况下选中的优化器会是 pooled。

我们不去讨论 none 和 legacy-hilo，前者不应该使用，后者的名字看上去像是古董。剩下 hilo、pooled 和 pooled-lo 其实是同一种算法，它们的区别在于

主键生成辅助表的数值。

Optimizer 究竟在表中记录了什么？

在表 3 中提到 hilo 优化器在辅助表中的数值是 bucket 的序号。这里 bucket 可以翻译成“桶”，也可翻译成“块”，其含义就是一段连续可分配的整数，如：1-10，50-100 等。桶的容量即是 increment_size 的值，假定 increment_size 的值为 50，那么桶的序号和每个桶容纳的整数可参看下表：

表 4. 桶的序号和容纳的整数

hilo 优化器把桶的序号放在了数据库辅助表中，pooled-lo 优化器把下一个桶的第一个整数放在数据库辅助表中，而 pooled 优化器则把下下桶的第一个整数放在数据库辅助表中。举个例子，如果 increment_size=50，当前某实体分到的主键编号为 60，可以推测出各个优化器及对应的数据库辅助表中的值。如下表所示：

序号	1	2	3	4	5	6	…
容纳的整数	1 -50	51 -100	101 -150	151 -200	201 -250	251 -300	…

表 5. 优化器与辅助表的值

优化器名称	当前主键 60，对应辅助表中的值
hilo	2

pooled-lo	101
pooled	151

一般来说，pooled-lo 比 pooled 更符合人的习惯，没有设置 hibernate.id.optimizer.pooled.prefer_lo 为 true 时，数据库辅助表的值会出乎人的意料。

程序员看到英文单词“pooled”，会和连接池这样的概念联系在一起，这里的池不过是一堆可用于主键分配的整数的“池”，其含义与连接池很相似。

新 TableGenerator 实例

最后，演示一下 Hibernate 新 TableGenerator 的完整功能。新 TableGenerator 的一些功能不在 JPA 中，因此不能使用 JPA 的 @TableGenerator 注解，而是要使用 Hibernate 自身的 @GenericGenerator 注解。

@GenericGenerator 注解有个 strategy 参数，用来指定主键生成器的名称或类名，类名是容易找到的，不过写起来太不方便了。生成器的名称却不大好找，翻遍 Hibernate 的 manual，devguide，都无法找到这些生成器的名称，最后还得去看源码。可以在 DefaultIdentifierGeneratorFactory 类中找到新 TableGenerator 的名称应该是“enhanced-table”。配置新 TableGenerator 的例子参看清单 4 的代码：



清单 4. 配置新 TableGenerator 的代码

```
1                @Entity @Table(name="emp4")
```

```
2               public class Employee4 {
3                   @GenericGenerator( name="id_gen",
4           strategy="enhanced-table",
5                   parameters = {
6                       @Parameter( name = "table_name", value =
7           "enhanced_gen"),
8                       @Parameter( name ="value_column_name", value =
9           "next"),
10                      @Parameter( name = "segment_column_name",value =
11          "segment_name"),
12                      @Parameter( name = "segment_value", value =
13          "emp_seq"),
14                      @Parameter( name = "increment_size", value = "10"),
15                      @Parameter( name = "optimizer",value = "pooled-lo")
16                  })
                    @Id @GeneratedValue(generator="id_gen")
                    private long id;
                    private String firstName; private String lastName;
                    ......
                    }
```

结束语

Hibernate 发展到现在，文档的更新有些落后于代码的实现，像新 Tablenerator 这样的特性，影响极大，很多程序员都在不清楚这些特性的情况下使用 Hibernate，这会给将来的升级带来隐患。Hibernate 官方的文档有待进一步完善，本文希望能够在这里做些补缺的工作。

原文链接：http://blog.sae.sina.com.cn/archives/2668

服务好“最后一公里”，高效 CDN 架构经验

国内，随着互联网的高速发展，因为各大通信公司的政策，造成了南电信北联通互通有局限性，再加上大小且质量参差不齐的运营商，在这特殊的氛围的互联互通下号称“八线合一”的机房开始崭露头角。互联网的广泛性使得网民分散在全国各地，由于全国地区的经济发展和互联网建设的不平衡，实际网民的体验往往受限于最后一公里的速度。在技术大喷井的年代，一些无聊或者有目的黑客攻击也开始涌现，无论是渗透还是 DDoS 攻击都非常频繁，时刻威胁着网站的安全……

上述种种问题，作为应用服务提供商，我们要如何解决此类问题呢？归根结底就是要充分利用好 CDN（Content Delivery Network，即内容分发网络）。

CDN 作用

缓存代理

缓存代理类似内容提供商源数据中心的一个透明镜像，这些内容可以在边缘服务器中缓存和分发，对于普通的网络用户来讲，它通过智能 DNS 的筛选，用户的请求被透明地指向离他最近的省内骨干节点，最大限度的缩短用户信息的传输距离。在任何时间、地点或者不同的运营商之间（尤其在中国），快速响应用户请求。

它是通过在网络各处放置节点服务器，所以无需更改源站的网络拓扑，而是根据智能路由和用户就近原则匹配，从而确保了内容快又稳定的传输，大大提高了用户访问网站的响应速度。

路由加速

CDN 服务初衷是确保快速可靠地分发静态内容，相对于动态内容来说，由于动态内容必须长连接来操持连接和通讯，只是用户到服务商之间的链路和质量都无法控制。因此为了提供快速的网络体验，有必要事先设置一些最佳路由。如省内骨干网，双线机房，以改善用户的网络体验。在中国典型的互联互通问题上，网络游戏加速就是一些最佳实践。

安全防护

利用好了 CDN 网络，无论面对是渗透还是 DDoS 攻击，攻击的目标大都会被

指向到了 CDN，进而保护了用户源站。因为 CDN 是分布式的，所以即使遭受 DDoS 攻击，也具备分散性，大大减少了源站收到毁灭打击的可能性。在架构的前期，还可以通过 CDN 做一些前置的安全保护工作，如拦截 SQL 注入、XSS 跨站、网站挂马、篡改等黑客攻击。

节省成本

CDN 节点机房只需要在当地运营商的单线机房，或者带宽相对便宜的城市，采购成本低。由于通过 CDN 减轻了源站压力，节点越多，源站面对任何时间高峰时的带宽峰值会被平均拉低。从而降低了后端服务器硬件规模和带宽的采购成本。 由于源站服务器规模的减少，后期运维成本也大大减少，可谓是一举多得。

由此可见，为了能够满足全国乃至世界各地和多线路运营商的不同用户都有最好的体验，构建 CDN 的分布式服务其重要性不言而喻。但是，在面对如何根据自身场景去设计一个 CDN 架构，或者如何选择以一个适合自己 CDN 服务提供商，这里面也有许多问题需要考量。

CDN 架构

存储介质 vs IO 的关系

这里先简单的介绍一下 SSD 介质的一些考量。SSD 作为采用电子存储介质进行数据存储和读取的一种技术，突破了传统机械硬盘的性能瓶颈，固态硬盘的全集成电路化、无任何机械运动部件的革命性设计，拥有极高的读取性能。

此环节，基本上不需要与传统的 SATA、SAS 作性能上的比较，SSD 的胜出毫无悬念。而在整体方案中，只需要考虑承受的价格、容量大小（如 120GB，160GB，300GB 等规格）、是否能够满足设计需求这些问题。

作者建议：如果允许， 能使用 SSD，就一定要考虑采用，用空间换性能，提升非常明显。

这里给几个 SSD 实战的小贴士：

选择 EXT4 文件系统+TRIM 模式（mount -o defaults,noatime,nodiratime,barrier=0,discard），Btrfs 建议少冒险

如果是使用三星的固态硬盘，可以尝试它贡献给开源的针对固态硬盘优化的 F2FS 文件系统，相当不错的选择

I/O Schedulers 调度算法，可以使用 CFQ 或者 Deadline 算法

内核参数调整，SSD 所在硬盘，echo 0 > /sys/block/sda/queue/rotational

随机读写 vs 顺序读写

机械硬盘的连续读写性很好，但随机读写性能很差。这是因为磁头移动至正确的磁道上需要时间，随机读写时，也就需要磁头和探针频繁的转动，而机械结构的磁头和探针的位置调整是十分费时的，这就严重影响到硬盘的寻址速度，进而影响到随机写入速度。

在存储小文件（图片）、OLTP 数据库应用时，随机读写性能（IOPS）是最重要指标。由于固态硬盘没有普通硬盘的机械结构，也不存在机械硬盘的寻道问题，因此系统能够在低于 1ms 的时间内对任意位置存储单元完成输入/输出操作。

作者经验笔记：

BIOS 里务必开启 AHCI 模式（能支持 SATA 热插拔和 NCQ 寻址方式，提速→300%，当然内核也要支持 AHCI 模式）

SSD 的主控芯片相当于大脑中枢，非常重要，建议用 Intel、Samsung、Marvell 等知名品牌

SSD 更适合应用在随机读写场景，因此需要认真思考什么场合应用

大文件 vs 小文件

大多数的存储系统都是针对大文件而设计的，对小文件而言，大文件的存储系统无法适应小文件的存储需求，它造成元数据管理、数据布局和 I/O 管理、Cache 管理、网络开销等方面性能和存储效率降低。

而且，文件系统的 inode 是线性存储的，因此，我们遍历一个目录下的文件，需要读取的磁盘的位置是来回跳跃的。不连续的读取意味着磁盘要不断的进行寻道，那么性能自然可想而知。

作者经验笔记：

无论大小文件，首选 EXT4 文件系统，Reiserfs/Btrfs 不要轻易尝试（虽然 B-tree 设计先进）

EXT4 针对小文件有所改进，使用了 inode 预分配，这使得 inode 具有很好的局部性特征，同一目录文件 inode 尽量放在一起，加速了目录寻址与操作性能。

EXT4 针对大文件使用了 extent/delay/multi 的数据块分配策略。这些策略使得大文件的数据块保持连续存储在磁盘上，数据寻址次数大大减少，显著提高 I/O

吞吐量。

XFS 在大文件方面，表现得不错，可以使用。

SSD 尽量应用在随机小文件读写的应用场景，毕竟容量宝贵，在有限的空间保存更多的文件是个明智之选。

有开发实力的可以选用基于 LevelDB 或其它的 KV 存储作底层文件系统，此为后话。

硬件红利 vs 软件设计

随着时间的推移，硬件升级已经突破了摩尔定律，在硬件不断升级带来的红利下，我们从最初的双核到四核、六核、八核心&超线程，从 2G、4G 内存到 8G、16G 甚至 128G 内存的情况下，同样的价格所带来的硬件升级，性能提升也是非常可观的，因此，设置合适的硬件淘汰时间点也很重要，当老旧服务器超过 3～5 年的服役期，务必考虑做新陈代谢式的升级，充分利用好硬件潜力，保证架构设计平滑有序稳定的升级。

反观软件设计，相对硬件升级，可谈的话题就比较多了，举个反例：比如说 Squid 软件的缺点（当然，诞生于 1996 年的 Squid 与 Apache 同样的古老，昔日的时代也是立下了汗马功劳，但时代进步就不能固步自封必须考虑革新）：

无法利用多核优势，造成单核 CPU 压力太高；

鸡肋的 DNS 进程必须要运行；

无法利用大内存做缓存加速；

COSS 设计上的先天缺陷，初始化甚至重启后重建索引慢；

偶然机器重启，修复的效率非常漫长，慢到让人崩溃；

更多详情参考：

Varnish Cache 的架构笔记，为什么一些古老的软件正在被新的设计思想所淘汰，如 Nginx 替代 Apache，ATS 替代 Squid，Postfix 替代 Sendmail 等等。

建议：

负载均衡技术应用得当，如 haproxy、lvs。一方面可以互援互备，另一方面也可以方便轮流升级；

要尝试新的软件开发思路和网络模型，如 epoll、aio、内存加速，连接复用和事件驱动机制；

系统优化

系统服务精简瘦身；

文件系统性能调优；

提高磁盘 IO 性能；

优化网络性能；

优化路由策略；

数据库的优化；

……这里就不展开详述了，以后有机会再介绍。

CDN 开源

开源世界里能够担当反向代理及缓存的软件不少，而且各有优劣。在这里，我就不一一介绍每个软件的介绍了，大家可以自行参考相关链接了解。

CDN 架构上要充分体现出抗攻击能力和灵活应变的原则。因此，我们将 CDN 节点分解成反向代理+缓存加速+攻击防御这三个不同层次的功能结构。

反向代理功能（作用：路由加速，隐藏主节点，负载均衡）

缓存加速功能（作用：静态推送，节省后端主节点带宽）

攻击防御功能（作用：快速解析，匹配过滤恶意攻击）

作为一个架构师，就必须要考虑如何选型，我们从性能、功能、配置上来进行比较筛选。

软件名称	性能	功能	过滤规则配置
Squid	不能多核是硬伤； 磁盘缓存容量有优势； 性能中等	多； 支持 ACL 角色控制； 支持 ICP 缓存协议	支持外部文件读取及热加载； 支持热启动
Varnish	多核支持； 内存缓存； 性能强	够用； 支持集群，但不支持 ICP 集群； 支持后端存活检查	不支持外部文件读取； 需要转义； 支持热启动

Nginx	多核支持； 支持代理插件； 性能较强	多； 支持集群，但不支持 ICP 集群； 支持后端存活检查； 通过插件可以充当多角色服务器	不支持外部文件读取； 需要转义； 支持热启动
Apache TS	多核支持； 磁盘/内存缓存； 性能强	够用； 支持后端存活检查； 支持 ICP 协议，Cluster 不稳定； 支持插件开发；	支持外部规则文件读取及热加载； 支持热启动
HAProxy	多核支持； 无缓存； 支持 HTTP 头部解析； 性能强	少，只专注 HTTP 头部解析和转发功能； 支持 ACL 角色控制； 支持后端存活检查	支持外部规则文件读取及热加载； 支持热启动； 支持会话粘滞和长连接

现在，我们对这三层功能结构充分了解，在测试调优及生产线的实践检验中，我们发现：

HTTP 防御性能：HAProxy 在应对大流量 CC 攻击时，做正则匹配及头部过滤时，CPU 消耗只占 10%~20%。其它软件均狂占 CPU 资源约 90%以上，容易成瓶颈导致整个系统无响应。

反向代理性能：单纯转发效率以内存缓存型的 Varnish 性能最强，ATS 和 Nginx 次之，考虑大容量缓存因素，ATS 也是个不错的选择。Nginx 是专门针对 C10K 的产物，性能不错，配合自己编写插件，业务可塑性很强。

过滤规则的可配置性：HAProxy，ATS，Squid 均支持规则文件读取、ACL 定制和热加载、热启动。Nginx 则不支持外部文件正则匹配，略差一点，但可塑性强。

负载均衡——高可用性:LVS

LVS 是个重量级、高效稳定的四层转发，虽然不能作七层 HTTP 协议的识别，但完全可以架设在七层之前，与上述的各种软件搭配使用。

所以，LVS 的使用并不会影响网络结构，后续仍然可以想上就上，前提是要兼顾到 LVS 的单点故障，这个我们可以通过 Keepalived/Heartbeat 来实现可用性和可靠性的保证。

原文链接：http://www.csdn.net/article/2014-02-11/2818377-how-to-design-cdn

探索推荐引擎内部的秘密

随着 Web 技术的发展，使得内容的创建和分享变得越来越容易。每天都有大量的图片、博客、视频发布到网上。信息的极度爆炸使得人们找到他们需要的信息将变得越来越难。传统的搜索技术是一个相对简单的帮助人们找到信息的工具，也广泛的被人们所使用，但搜索引擎并不能完全满足用户对信息发现的需求，原因一是用户很难用恰当的关键词描述自己的需求，二是基于关键词的信息检索在很多情况下是不够的。而推荐引擎的出现，使用户获取信息的方式从简单的目标明确的数据的搜索转换到更高级更符合人们使用习惯的上下文信息更丰富的信息发现。

本文将深入介绍推荐引擎的工作原理，和其中涉及的各种推荐机制，以及它们各自的优缺点和适用场景，帮助用户清楚的了解和快速构建适合自己的推荐引擎。

信息发现

如今已经进入了一个数据爆炸的时代，随着 Web 2.0 的发展， Web 已经变成数据分享的平台，那么，如何让人们在海量的数据中想要找到他们需要的信息将变得越来越难。

在这样的情形下，搜索引擎（Google，Bing，百度等等）成为大家快速找到

目标信息的最好途径。在用户对自己需求相对明确的时候，用搜索引擎很方便的通过关键字搜索很快的找到自己需要的信息。但搜索引擎并不能完全满足用户对信息发现的需求，那是因为在很多情况下，用户其实并不明确自己的需要，或者他们的需求很难用简单的关键字来表述。又或者他们需要更加符合他们个人口味和喜好的结果，因此出现了推荐系统，与搜索引擎对应，大家也习惯称它为推荐引擎。

随着推荐引擎的出现，用户获取信息的方式从简单的目标明确的数据的搜索转换到更高级更符合人们使用习惯的信息发现。

如今，随着推荐技术的不断发展，推荐引擎已经在电子商务（E-commerce，例如 Amazon，当当网）和一些基于 social 的社会化站点（包括音乐，电影和图书分享，例如豆瓣，Mtime 等）都取得很大的成功。这也进一步的说明了，Web2.0 环境下，在面对海量的数据，用户需要这种更加智能的，更加了解他们需求，口味和喜好的信息发现机制。

推荐引擎

前面介绍了推荐引擎对于现在的 Web2.0 站点的重要意义，这一章我们将讲讲推荐引擎到底是怎么工作的。推荐引擎利用特殊的信息过滤技术，将不同的物品或内容推荐给可能对它们感兴趣的用户。

图 1. 推荐引擎工作原理图

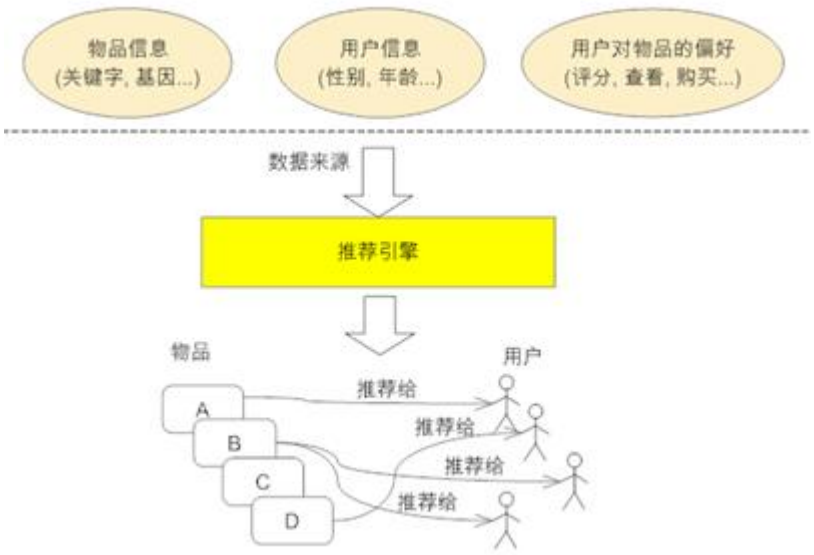


图 1 给出了推荐引擎的工作原理图，这里先将推荐引擎看作黑盒，它接受的输入是推荐的数据源，一般情况下，推荐引擎所需要的数据源包括：

要推荐物品或内容的元数据，例如关键字，基因描述等；

系统用户的基本信息，例如性别，年龄等

用户对物品或者信息的偏好，根据应用本身的不同，可能包括用户对物品的评分，用户查看物品的记录，用户的购买记录等。其实这些用户的偏好信息可以分为两类：

显式的用户反馈：这类是用户在网站上自然浏览或者使用网站以外，显式的提供反馈信息，例如用户对物品的评分，或者对物品的评论。

隐式的用户反馈：这类是用户在使用网站是产生的数据，隐式的反应了用户对物品的喜好，例如用户购买了某物品，用户查看了某物品的信息等等。

显式的用户反馈能准确的反应用户对物品的真实喜好，但需要用户付出额外的代价，而隐式的用户行为，通过一些分析和处理，也能反映用户的喜好，只是数据不是很精确，有些行为的分析存在较大的噪音。但只要选择正确的行为特征，隐式的用户反馈也能得到很好的效果，只是行为特征的选择可能在不同的应用中有很大的不同，例如在电子商务的网站上，购买行为其实就是一个能很好表现用户喜好的隐式反馈。

推荐引擎根据不同的推荐机制可能用到数据源中的一部分，然后根据这些数据，分析出一定的规则或者直接对用户对其他物品的喜好进行预测计算。这样推荐引擎可以在用户进入的时候给他推荐他可能感兴趣的物品。

推荐引擎的分类

推荐引擎的分类可以根据很多指标，下面我们一一介绍一下：

推荐引擎是不是为不同的用户推荐不同的数据根据这个指标，推荐引擎可以分为基于大众行为的推荐引擎和个性化推荐引擎

根据大众行为的推荐引擎，对每个用户都给出同样的推荐，这些推荐可以是静态的由系统管理员人工设定的，或者基于系统所有用户的反馈统计计算出的当下比较流行的物品。

个性化推荐引擎，对不同的用户，根据他们的口味和喜好给出更加精确的推荐，这时，系统需要了解需推荐内容和用户的特质，或者基于社会化网络，通过

找到与当前用户相同喜好的用户，实现推荐。

这是一个最基本的推荐引擎分类，其实大部分人们讨论的推荐引擎都是将个性化的推荐引擎，因为从根本上说，只有个性化的推荐引擎才是更加智能的信息发现过程。

根据推荐引擎的数据源其实这里讲的是如何发现数据的相关性，因为大部分推荐引擎的工作原理还是基于物品或者用户的相似集进行推荐。那么参考图 1 给出的推荐系统原理图，根据不同的数据源发现数据相关性的方法可以分为以下几种：

根据系统用户的基本信息发现用户的相关程度，这种被称为基于人口统计学的推荐（Demographic-based Recommendation）

根据推荐物品或内容的元数据，发现物品或者内容的相关性，这种被称为基于内容的推荐（Content-based Recommendation）

根据用户对物品或者信息的偏好，发现物品或者内容本身的相关性，或者是发现用户的相关性，这种被称为基于协同过滤的推荐（Collaborative Filtering-based Recommendation）。

根据推荐模型的建立方式可以想象在海量物品和用户的系统中，推荐引擎的计算量是相当大的，要实现实时的推荐务必需要建立一个推荐模型，关于推荐模型的建立方式可以分为以下几种：

基于物品和用户本身的，这种推荐引擎将每个用户和每个物品都当作独立的实体，预测每个用户对于每个物品的喜好程度，这些信息往往是用一个二维矩阵描述的。由于用户感兴趣的物品远远小于总物品的数目，这样的模型导致大量的数据空置，即我们得到的二维矩阵往往是一个很大的稀疏矩阵。同时为了减小计算量，我们可以对物品和用户进行聚类， 然后记录和计算一类用户对一类物品的喜好程度，但这样的模型又会在推荐的准确性上有损失。

基于关联规则的推荐（Rule-based Recommendation）：关联规则的挖掘已经是数据挖掘中的一个经典的问题，主要是挖掘一些数据的依赖关系，典型的场景就是“购物篮问题”，通过关联规则的挖掘，我们可以找到哪些物品经常被同时购买，或者用户购买了一些物品后通常会购买哪些其他的物品，当我们挖掘出这些关联规则之后，我们可以基于这些规则给用户进行推荐。

基于模型的推荐（Model-based Recommendation）：这是一个典型的机器学习的问题，可以将已有的用户喜好信息作为训练样本，训练出一个预测用户喜好的模型，这样以后用户在进入系统，可以基于此模型计算推荐。这种方法的问题在于如何将用户实时或者近期的喜好信息反馈给训练好的模型，从而提高推荐的准确度。

其实在现在的推荐系统中，很少有只使用了一个推荐策略的推荐引擎，一般都是在不同的场景下使用不同的推荐策略从而达到最好的推荐效果，例如 Amazon 的推荐，它将基于用户本身历史购买数据的推荐，和基于用户当前浏览的物品的推荐，以及基于大众喜好的当下比较流行的物品都在不同的区域推荐给用户，让用户可以从全方位的推荐中找到自己真正感兴趣的物品。

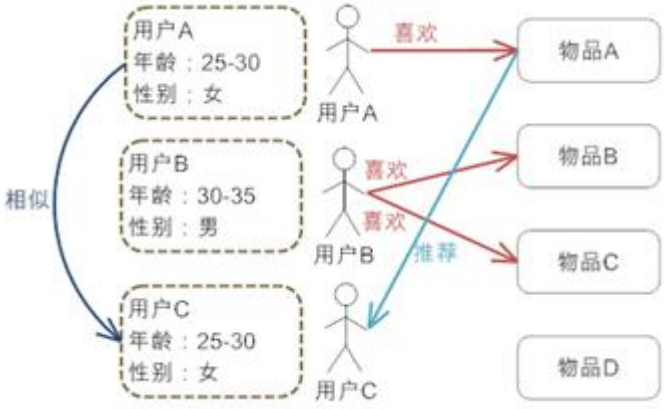
深入推荐机制

这一章的篇幅，将详细介绍各个推荐机制的工作原理，它们的优缺点以及应用场景。

基于人口统计学的推荐

基于人口统计学的推荐机制（Demographic-based Recommendation）是一种最易于实现的推荐方法，它只是简单的根据系统用户的基本信息发现用户的相关程度，然后将相似用户喜爱的其他物品推荐给当前用户，图 2 给出了这种推荐的工作原理。

图 2. 基于人口统计学的推荐机制的工作原理



从图中可以很清楚的看到，首先，系统对每个用户都有一个用户 Profile 的建模，其中包括用户的基本信息，例如用户的年龄，性别等等；然后，系统会根据用户的 Profile 计算用户的相似度，可以看到用户 A 的 Profile 和用户 C 一

样，那么系统会认为用户 A 和 C 是相似用户，在推荐引擎中，可以称他们是“邻居”；最后，基于“邻居”用户群的喜好推荐给当前用户一些物品，图中将用户 A 喜欢的物品 A 推荐给用户 C。

这种基于人口统计学的推荐机制的好处在于：

因为不使用当前用户对物品的喜好历史数据，所以对于新用户来讲没有“冷启动（Cold Start）”的问题。

这个方法不依赖于物品本身的数据，所以这个方法在不同物品的领域都可以使用，它是领域独立的（domain-independent）。

那么这个方法的缺点和问题是什么呢？这种基于用户的基本信息对用户进行分类的方法过于粗糙，尤其是对品味要求较高的领域，比如图书，电影和音乐等领域，无法得到很好的推荐效果。可能在一些电子商务的网站中，这个方法可以给出一些简单的推荐。另外一个局限是，这个方法可能涉及到一些与信息发现问题本身无关却比较敏感的信息，比如用户的年龄等，这些用户信息不是很好获取。

基于内容的推荐

基于内容的推荐是在推荐引擎出现之初应用最为广泛的推荐机制，它的核心思想是根据推荐物品或内容的元数据，发现物品或者内容的相关性，然后基于用户以往的喜好记录，推荐给用户相似的物品。图 3 给出了基于内容推荐的基本原理。

图 3. 基于内容推荐机制的基本原理

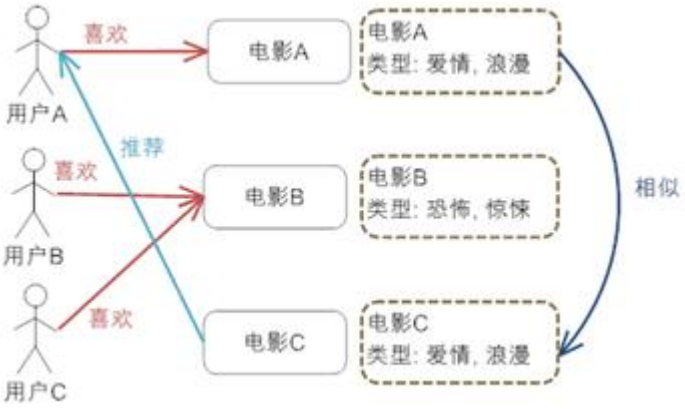


图 3 中给出了基于内容推荐的一个典型的例子，电影推荐系统，首先我们需要对电影的元数据有一个建模，这里只简单的描述了一下电影的类型；然后通

过电影的元数据发现电影间的相似度，因为类型都是“爱情，浪漫”电影 A 和 C 被认为是相似的电影（当然，只根据类型是不够的，要得到更好的推荐，我们还可以考虑电影的导演，演员等等）；最后实现推荐，对于用户 A，他喜欢看电影 A，那么系统就可以给他推荐类似的电影 C。

这种基于内容的推荐机制的好处在于它能很好的建模用户的口味，能提供更加精确的推荐。但它也存在以下几个问题：

需要对物品进行分析和建模，推荐的质量依赖于对物品模型的完整和全面程度。在现在的应用中我们可以观察到关键词和标签（Tag）被认为是描述物品元数据的一种简单有效的方法。

物品相似度的分析仅仅依赖于物品本身的特征，这里没有考虑人对物品的态度。

因为需要基于用户以往的喜好历史做出推荐，所以对于新用户有“冷启动”的问题。

虽然这个方法有很多不足和问题，但他还是成功的应用在一些电影，音乐，图书的社交站点，有些站点还请专业的人员对物品进行基因编码，比如潘多拉，在一份报告中说道，在潘多拉的推荐引擎中，每首歌有超过 100 个元数据特征，包括歌曲的风格，年份，演唱者等等。

基于协同过滤的推荐

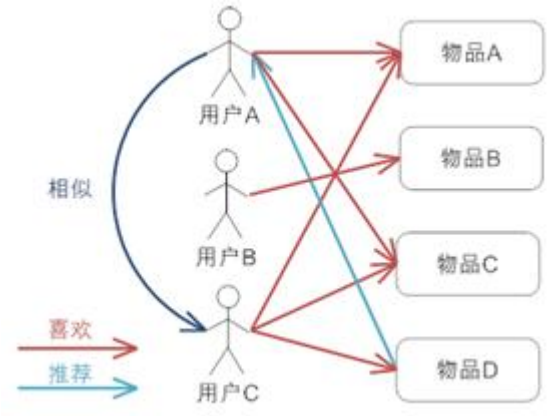
随着 Web2.0 的发展，Web 站点更加提倡用户参与和用户贡献，因此基于协同过滤的推荐机制因运而生。它的原理很简单，就是根据用户对物品或者信息的偏好，发现物品或者内容本身的相关性，或者是发现用户的相关性，然后再基于这些关联性进行推荐。基于协同过滤的推荐可以分为三个子类：基于用户的推荐（User-based Recommendation），基于项目的推荐（Item-based Recommendation）和基于模型的推荐（Model-based Recommendation）。下面我们一个一个详细的介绍着三种协同过滤的推荐机制。

基于用户的协同过滤推荐

基于用户的协同过滤推荐的基本原理是，根据所有用户对物品或者信息的偏好，发现与当前用户口味和偏好相似的“邻居”用户群，在一般的应用中是采用计算“K- 邻居”的算法；然后，基于这 K 个邻居的历史偏好信息，为当前用户进行

推荐。下图 4 给出了原理图。

图 4. 基于用户的协同过滤推荐机制的基本原理



上图示意出基于用户的协同过滤推荐机制的基本原理，假设用户 A 喜欢物品 A，物品 C，用户 B 喜欢物品 B，用户 C 喜欢物品 A ，物品 C 和物品 D；从这些用户的历史喜好信息中，我们可以发现用户 A 和用户 C 的口味和偏好是比较类似的，同时用户 C 还喜欢物品 D，那么我们可以推断用户 A 可能也喜欢物品 D，因此可以将物品 D 推荐给用户 A。

基于用户的协同过滤推荐机制和基于人口统计学的推荐机制都是计算用户的相似度，并基于“邻居”用户群计算推荐，但它们所不同的是如何计算用户的相似度，基于人口统计学的机制只考虑用户本身的特征，而基于用户的协同过滤机制可是在用户的历史偏好的数据上计算用户的相似度，它的基本假设是，喜欢类似物品的用户可能有相同或者相似的口味和偏好。

基于项目的协同过滤推荐

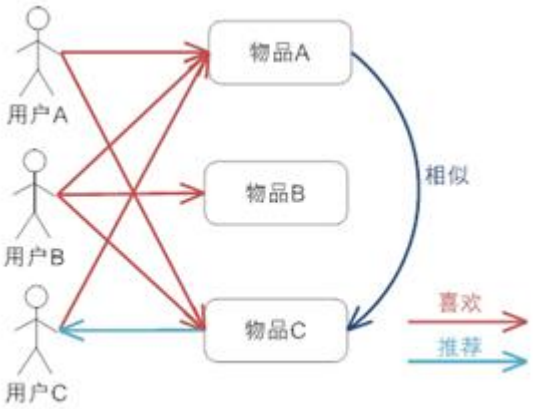
基于项目的协同过滤推荐的基本原理也是类似的，只是说它使用所有用户对物品或者信息的偏好，发现物品和物品之间的相似度，然后根据用户的历史偏好信息，将类似的物品推荐给用户，图 5 很好的诠释了它的基本原理。

假设用户 A 喜欢物品 A 和物品 C，用户 B 喜欢物品 A，物品 B 和物品 C，用户 C 喜欢物品 A，从这些用户的历史喜好可以分析出物品 A 和物品 C 时比较类似的，喜欢物品 A 的人都喜欢物品 C，基于这个数据可以推断用户 C 很有可能也喜欢物品 C，所以系统会将物品 C 推荐给用户 C。

与上面讲的类似，基于项目的协同过滤推荐和基于内容的推荐其实都是基于

物品相似度预测推荐，只是相似度计算的方法不一样，前者是从用户历史的偏好推断，而后者是基于物品本身的属性特征信息。

图 5. 基于项目的协同过滤推荐机制的基本原理



同时协同过滤，在基于用户和基于项目两个策略中应该如何选择呢？其实基于项目的协同过滤推荐机制是 Amazon 在基于用户的机制上改良的一种策略，因为在大部分的 Web 站点中，物品的个数是远远小于用户的数量的，而且物品的个数和相似度相对比较稳定，同时基于项目的机制比基于用户的实时性更好一些。但也不是所有的场景都是这样的情况，可以设想一下在一些新闻推荐系统中，也许物品，也就是新闻的个数可能大于用户的个数，而且新闻的更新程度也有很快，所以它的形似度依然不稳定。所以，其实可以看出，推荐策略的选择其实和具体的应用场景有很大的关系。

基于模型的协同过滤推荐

基于模型的协同过滤推荐就是基于样本的用户喜好信息，训练一个推荐模型，然后根据实时的用户喜好的信息进行预测，计算推荐。

基于协同过滤的推荐机制是现今应用最为广泛的推荐机制，它有以下几个显著的优点：

它不需要对物品或者用户进行严格的建模，而且不要求物品的描述是机器可理解的，所以这种方法也是领域无关的。

这种方法计算出来的推荐是开放的，可以共用他人的经验，很好的支持用户发现潜在的兴趣偏好

而它也存在以下几个问题：

方法的核心是基于历史数据，所以对新物品和新用户都有“冷启动”的问题。

推荐的效果依赖于用户历史偏好数据的多少和准确性。

在大部分的实现中，用户历史偏好是用稀疏矩阵进行存储的，而稀疏矩阵上的计算有些明显的问题，包括可能少部分人的错误偏好会对推荐的准确度有很大的影响等等。

对于一些特殊品味的用户不能给予很好的推荐。

由于以历史数据为基础，抓取和建模用户的偏好后，很难修改或者根据用户的使用演变，从而导致这个方法不够灵活。

混合的推荐机制

在现行的 Web 站点上的推荐往往都不是单纯只采用了某一种推荐的机制和策略，他们往往是将多个方法混合在一起，从而达到更好的推荐效果。关于如何组合各个推荐机制，这里讲几种比较流行的组合方法。

加权的混合（Weighted Hybridization）：用线性公式（linear formula）将几种不同的推荐按照一定权重组合起来，具体权重的值需要在测试数据集上反复实验，从而达到最好的推荐效果。

切换的混合（Switching Hybridization）：前面也讲到，其实对于不同的情况（数据量，系统运行状况，用户和物品的数目等），推荐策略可能有很大的不同，那么切换的混合方式，就是允许在不同的情况下，选择最为合适的推荐机制计算推荐。

分区的混合（Mixed Hybridization）：采用多种推荐机制，并将不同的推荐结果分不同的区显示给用户。其实，Amazon，当当网等很多电子商务网站都是采用这样的方式，用户可以得到很全面的推荐，也更容易找到他们想要的东西。

分层的混合（Meta-Level Hybridization）：采用多种推荐机制，并将一个推荐机制的结果作为另一个的输入，从而综合各个推荐机制的优缺点，得到更加准确的推荐。

推荐引擎的应用

介绍完推荐引擎的基本原理，基本推荐机制，下面简要分析几个有代表性的推荐引擎的应用，这里选择两个领域：Amazon 作为电子商务的代表，豆瓣作为社交网络的代表。

推荐在电子商务中的应用 – Amazon

Amazon 作为推荐引擎的鼻祖，它已经将推荐的思想渗透在应用的各个角落。Amazon 推荐的核心是通过数据挖掘算法和比较用户的消费偏好于其他用户进行对比，借以预测用户可能感兴趣的商品。对应于上面介绍的各种推荐机制，Amazon 采用的是分区的混合的机制，并将不同的推荐结果分不同的区显示给用户，图 6 和图 7 展示了用户在 Amazon 上能得到的推荐。

图 6. Amazon 的推荐机制 – 首页

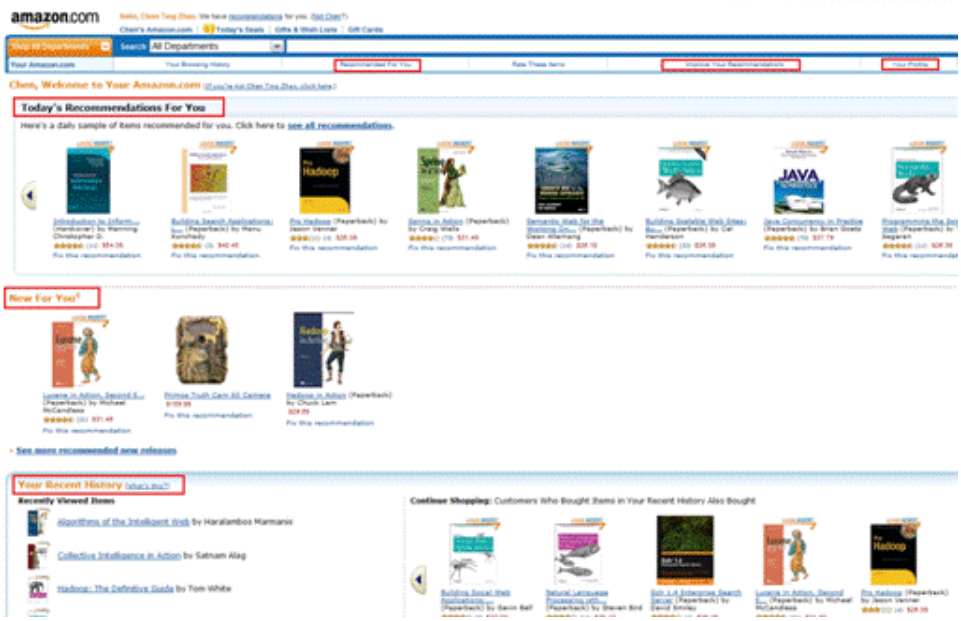
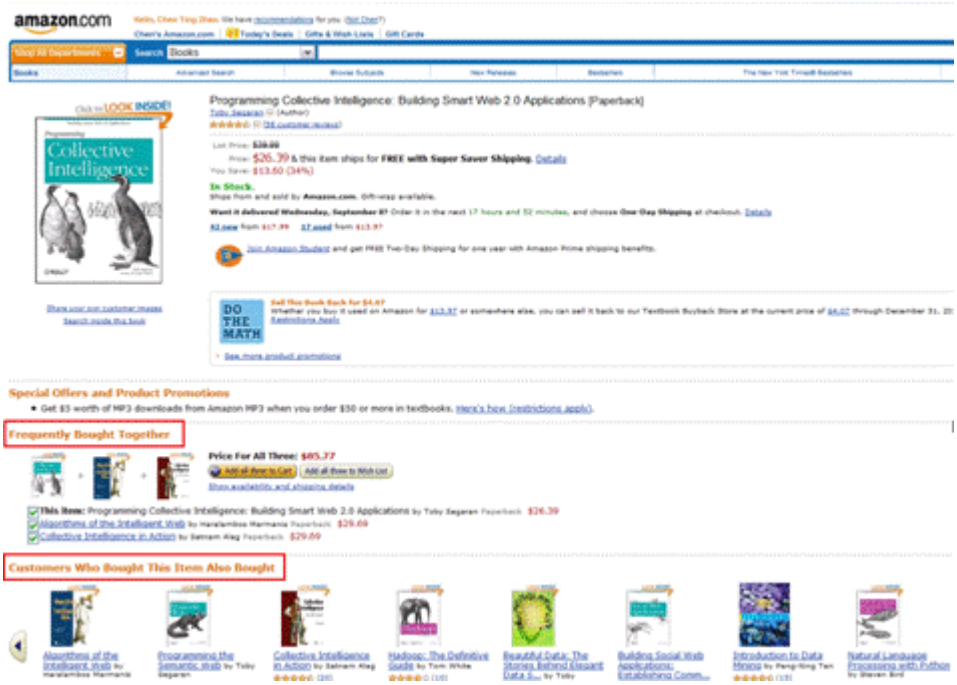


图 7. Amazon 的推荐机制 – 浏览物品



Amazon 利用可以记录的所有用户在站点上的行为，根据不同数据的特点对它们进行处理，并分成不同区为用户推送推荐：

今日推荐 (Today's Recommendation For You): 通常是根据用户的近期的历史购买或者查看记录，并结合时下流行的物品给出一个折中的推荐。

新产品的推荐 (New For You): 采用了基于内容的推荐机制 (Content-based Recommendation)，将一些新到物品推荐给用户。在方法选择上由于新物品没有大量的用户喜好信息，所以基于内容的推荐能很好的解决这个"冷启动"的问题。

捆绑销售 (Frequently Bought Together): 采用数据挖掘技术对用户的购买行为进行分析，找到经常被一起或同一个人购买的物品集，进行捆绑销售，这是一种典型的基于项目的协同过滤推荐机制。

别人购买 / 浏览的商品 (Customers Who Bought/See This Item Also Bought/See): 这也是一个典型的基于项目的协同过滤推荐的应用，通过社会化机制用户能更快更方便的找到自己感兴趣的物品。

值得一提的是，Amazon 在做推荐时，设计和用户体验也做得特别独到：

Amazon 利用有它大量历史数据的优势，量化推荐原因。

基于社会化的推荐，Amazon 会给你事实的数据，让用户信服，例如：购买此物品的用户百分之多少也购买了那个物品；

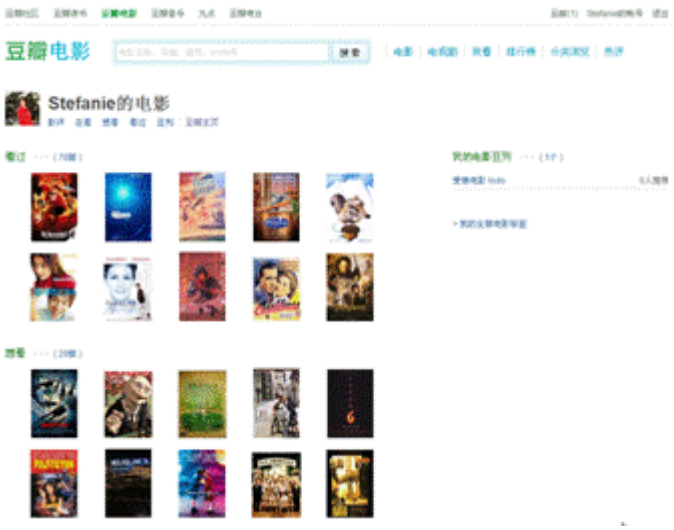
基于物品本身的推荐，Amazon 也会列出推荐的理由，例如：因为你的购物框中有 ***，或者因为你购买过 ***，所以给你推荐类似的 ***。

另外，Amazon 很多推荐是基于用户的 profile 计算出来的，用户的 profile 中记录了用户在 Amazon 上的行为，包括看了那些物品，买了那些物品，收藏夹和 wish list 里的物品等等，当然 Amazon 里还集成了评分等其他的用户反馈的方式，它们都是 profile 的一部分，同时，Amazon 提供了让用户自主管理自己 profile 的功能，通过这种方式用户可以更明确的告诉推荐引擎他的品味和意图是什么。

推荐在社交网站中的应用 – 豆瓣

豆瓣是国内做的比较成功的社交网站，它以图书，电影，音乐和同城活动为中心，形成一个多元化的社交网络平台，自然推荐的功能是必不可少的，下面我们看看豆瓣是如何推荐的。

图 8．豆瓣的推荐机制 – 豆瓣电影



当你在豆瓣电影中将一些你看过的或是感兴趣的电影加入你看过和想看的列表里，并为它们做相应的评分，这时豆瓣的推荐引擎已经拿到你的一些偏好信息，那么它将给你展示如图 8 的电影推荐。

图 9．豆瓣的推荐机制 – 基于用户品味的推荐



豆瓣的推荐是通过“豆瓣猜”，为了让用户清楚这些推荐是如何来的，豆瓣还给出了“豆瓣猜”的一个简要的介绍。

“你的个人推荐是根据你的收藏和评价自动得出的，每个人的推荐清单都不

同。你的收藏和评价越多，豆瓣给你的推荐会越准确和丰富。

每天推荐的内容可能会有变化。随着豆瓣的长大，给你推荐的内容也会越来越准。”

这一点让我们可以清晰明了的知道，豆瓣必然是基于社会化的协同过滤的推荐，这样用户越多，用户的反馈越多，那么推荐的效果会越来越准确。

相对于 Amazon 的用户行为模型，豆瓣电影的模型更加简单，就是“看过”和“想看”，这也让他们的推荐更加专注于用户的品味，毕竟买东西和看电影的动机还是有很大不同的。

另外，豆瓣也有基于物品本身的推荐，当你查看一些电影的详细信息的时候，他会给你推荐出“喜欢这个电影的人也喜欢的电影”， 如图 10，这是一个基于协同过滤的应用。

图 10．豆瓣的推荐机制 – 基于电影本身的推荐



总结

在网络数据爆炸的年代，如何让用户更快的找到想要的数据，如何让用户发现自己潜在的兴趣和需求，无论是对于电子商务还是社会网络的应用都是至关重要的。推荐引擎的出现，使得这个问题越来越被大家关注。但对大多数人来讲，也许还在惊叹它为什么总是能猜到你到底想要些什么。推荐引擎的魔力在于你不清楚在这个推荐背后，引擎到底记录和推理了些什么。

通过这篇综述性的文章，你可以了解，其实推荐引擎只是默默的记录和观察你的一举一动，然后再借由所有用户产生的海量数据分析和发现其中的规律，进而慢慢的了解你，你的需求，你的习惯，并默默的无声息的帮助你快速的解决你的问题，找到你想要的东西。

其实，回头想想，很多时候，推荐引擎比你更了解你自己。

通过第一篇文章，相信大家对推荐引擎有一个清晰的第一印象，本系列的下一篇文章将深入介绍基于协同过滤的推荐策略。在现今的推荐技术和算法中，最被大家广泛认可和采用的就是基于协同过滤的推荐方法。它以其方法模型简单，数据依赖性低，数据方便采集，推荐效果较优等多个优点成为大众眼里的推荐算法“No.1”。本文将带你深入了解协同过滤的秘密，并给出基于 Apache Mahout 的协同过滤算法的高效实现。Apache Mahout 是 ASF 的一个较新的开源项目，它源于 Lucene，构建在 Hadoop 之上，关注海量数据上的机器学习经典算法的高效实现。

原文链接：http://blog.sae.sina.com.cn/archives/2706

一起 select 引起的崩溃

2 月 13 日，陌陌争霸 计划在下午 16:00 例行维护，之前已经稳定运行了很长时间了。没想到在 14:30 左右，有一台从机意外崩溃。再次之前，从机崩溃并不会引起系统坏死，只需要新启动一台从机即可。但这次似乎不一样，所有玩家均不能登陆游戏，不得已，提前进行了长达两小时的紧急维护（将例行维护的工作合并）。

这次从机崩溃进行的大规模事故，可简单描述为：当从机上 5K 玩家掉线后，引起这批用户立刻连接其它从机。而事故发生后另一个小 bug 导致没有及时从系统中把这批用户重置，导致这批用户不可能立刻重新进入系统，反复重试。而每台从机上的网关配置最大连接数过小，导致他们阻塞了其他用户的登陆。

维护后修正了后面几个问题，由于时间紧迫，没有细查崩溃的直接原因。（另一个小插曲是，core 文件被输出到一个容量很小的分区，分区容量不足，导致 core 文件被截断，增加了分析难度。）系统重开后，不到半小时又有一台从机崩溃，这次没有产生雪崩，立刻启用了备用机替换。可在 5 分钟后，新的一起崩

溃事件在另一台从机上发生。这使我们意识到问题的严重性。

由于后来的崩溃拿到了完整的 core 文件，可以看到 C 层的崩溃发生在游戏服务器和我们的运营平台服务器（分属两个机房）的对接环节。由于机房间网络状态不稳定，发生了一次重连操作，再重新建立连接时，程序出错。但是程序调用栈已经被部分破坏。同时，错误 log 中有大量玩家领奖品失败的记录 。最快速的判断是，两者有一定的相关性。我们已经两个月没有修改 C 代码了，之前一直很稳定，过年期间无人守护都没有发生这种严重的事故，所以很难理解 bug 的成因。最近只在 lua 曾增加了一些业务逻辑，比如一些奖品发放。不太能确定到底是何为因何为果。

到了 13 日晚饭时间，依然没有头绪。为了减少晚上再次崩溃的可能，我决定关闭奖品领取模块，再慢慢追查 bug 。

core 文件中的调用栈破坏使得从现场做分析变得有些困难。所以按一般的 bug 追查策略，我们推测是由于这次发奖的某种流程导致的问题。结合三次崩溃的现象，大致认为是某个玩家领取奖励触发了某个很难测试到的分支。花了一点时间在三次崩溃前 20 分钟的 log 中获得了几万个玩家的 id 。放在一起比对，只有一个玩家恰好出现在三个名单中。

但事后表明，这个玩家的出现时间完全是巧合而已。我们没有从这个玩家的信息中获得线索。但三次崩溃都出现在游戏服务器和运营平台对接重连中，这使得这部分相关代码变得最为可疑。

写这块代码的蜗牛同学正在马来西亚度婚假，所以就不打搅他了。13 日晚上，我阅读了所有我们编写的平台对接代码，没有找到问题。弄得这一整晚，晓靖和我都非常不爽。

14 日，我改去查蜗牛用到的 rabbitmq 的 C 库 ，初看下来也没啥问题。

在毫无线索的情况下，只好开始在 core 文件中重新寻找蛛丝马迹。把未损坏的堆栈数据和反编译的代码逐个比对。发现在 amqp_open_socket_noblock 函数中 portnumber 这个局部数组全部为 0 ，但 sockfd 却不是 -1 。这说明 socket 被创建后，portnumber 数组却被覆盖掉了。

再仔细阅读才发现，原来是 rabbitmq-c 为了控制 connect 的超时时间，使用了非阻塞 connect ，然后使用 select 判断 fd 是否可写来判定连接是否建立。

select 调用未考虑 sockfd 大于 FD_SETSIZE 。linux 的 select 手册中有这样一行注释：

Anfd_set is a fixed size buffer. Executing FD_CLR() or FD_SET() with a value of fd that is negative or is equal to or larger than FD_SETSIZE will result in undefined behavior. Moreover, POSIX requires fd to be a valid file descriptor.

这是一个很容易被程序员忽视的坑。FD_SET 其实是一个位数组，linux 默认是 1024 bit 。而 FD_SET 只是简单的把 fd 当作一个序号按位向位数组写数据。select 应该也是如此。所以当 fd 大于 1024 时，就会写越界。正确的方法是当需要 select 的 fd 大于等于 1024 时，最好在堆上分配 FD_SET ，分配空间应至少保证 fd/8 + 1 以上；或者用 poll / WSAPoll 更好。

一般的程序不会有这个问题是因为通常一个进程需要的文件 fd 不会超过 1024 。而我们的系统管理了大量的外部连接，超过 1024 轻而易举。之前，机房间的网络正常，只在程序启动时创建连接一次连接。这时尚无外部连接，这个连接的 fd 一定是小于 1024 的，而之后也不会发生重连。

原文链接:http://blog.codingnow.com/2014/02/select_bug.html

课堂上传纸条如何防范中间人攻击？

首先第一种方案就是排除中间人，直接建立点对点连接，例如直接将纸条从空中抛送过去，顺便还能锻炼自己的投篮水平，为以后进入国家队作准备。缺点就是如果隔太远这种方式丢包率太大了。

第二种方案就是只使用可信网关连接，在自己与目标之间只请求自己信得过的哥们儿姐们儿进行传递。如果必须在恶劣的情况下进行安全的数据传递，那么有如下几个技巧：

1、CDMA（码分多址），将一段信息拆分成两份，再从不同渠道传递，这是非常简单的一种加密方式。例如把要说的话写成一行，再从中间横向撕开。目标

收到两份信息后，再拼接起来进行解密。其具有加密解密简便易行的优点。但是有一些字即使从中间撕开通过上下文也能判断出来大致意思。所以这还需要其他辅助加密方式，后面谈到。请注意 CDMA 技术完全不是这个意思，仅为搞笑

2、TD-SCDMA（时分同步码分多址），将一份信息拆分传递时，应当等目标安全收到一半信息后，再进行第二次传输。这样老师截获完整信息的可能性将大大降低。请注意 TD-SCDMA 技术完全不是这个意思，仅为搞笑

3、Scrambler（随机扰码），在字里行间插入一些不可解读的字符，例如原文是：放学后到学校后面的小树林等我，加入随机扰码后：放魅学后蒙到学赞校后翠面的小澧树林等齷我由于随机扰码的存在，即使得到一半的纸条，也很难通过上下文推测出原文。

4、跳板技术，有时候为了避免老师发现通信链路的存在，不应建立直接的链路，例如你要传到左上角的女孩手里，则应先向右传，通过多级代理跳板，让老师无法发现通信的源头和目标。

5、Token-Ring（令牌环网），事实上由于跳板技术需要很多人参与，有时候你无法说服所有人，这时候可以组成令牌环网，把需要传递纸条的男女同学组成一个令牌环网，按照既定的环路，有序的传输，例如一个典型的令牌环网像是这样，你是 A 需要传递消息给 G，A-B-C-D-E-F-G，也就是你先传给 B，然后 B 给 C，如此。同时 F 也有消息要传递给 B，则采用环路的形式，F-G-H-I-A-B，也就是说 F 的纸条传递给 G，G 再传递给 H，H 传给 I，I 传给你，你再将它给 B。这样整个令牌环网的每个节点都可以获益于环网的信息传递，老师完全无法得知纸条的源和目节点。此处仅利用了令牌环网的拓扑结构

6、MultiLink（多重链路），有时候链路的节点可能会临时性瘫痪（被老师叫起来回答问题）或者变得非常不安全（老师站在旁边），所以建立网络时任何两个节点之间不能只有唯一的链路，需要有多个备选的链路方案。

7、TTL（存活时间），在长时间的传输过程中，纸条被老师发现的可能性将增大。所以可以约定纸条如果在 N 个节点传递之后仍然无法抵达目标，则应由该节点直接销毁。声明：以上出现的专有名词，仅为博君一乐，不代表此种技术真正的实现原理和方式，切记切记。

原文链接：http://www.zhihu.com/question/22558998/answer/21803111

[程序人生]

《9Tech 访谈录》揭秘游戏王子白泽的技术之路

近日，9Tech 社区专访了 HTML5 游戏开发者社区联合创办人及玛尔斯公司创意总监白泽，他是一位热爱技术、乐于分享、拥抱开源的 HTML5 热爱者。但他背后还隐藏着一个身份——资深 Flash 开发者，以式神自称。他曾经从事过数年 AS 开发，并在 Flash 圈内小有名气。很多读者可能会好奇，Flash 与 HTML5 水火不容，各方开发者曾为此争论不休。而白泽为何会转向“敌方”阵营？本文将会详细解读白泽开发史，对 Flash 与 HTML5 的独特见解，移动网页游戏的发展趋势的预测，以及拥抱开源的态度和经验分享。



HTML5 游戏开发者社区联合创办人及玛尔斯公司创意总监 白泽

非技术出身 Flash 教会我融会贯通

2006 年，学生阶段的白泽对可以做动画的 Flash 产生了浓郁的兴趣，并开始尝试创作 Flash 动画。后因迫于压力及面子，从而每日强迫自己深入研究 Flash。当时，还未正经触碰编程，仅是表面的动画效果。随着时间的推移，白泽接触到一本 Flash 教程书籍，他被书中的按键监听器所吸引，从而萌发出制作游戏的想法。然而，当时白泽还只是一名美术生，未曾接触过编程，游戏算法对他而言更是天方夜谭。

但他却从美术的角度去理解游戏，通过不懈努力，自己创作出一款拥有背景

循环、控制移动、发射子弹的飞机类游戏。这一次的成功加深了白泽对游戏的兴趣，他开始自学编程。从最初的时间轴、帧，慢慢延伸到类、面向对象、游戏特效等。白泽为了完成自己开发游戏的梦想，2008 年真正开始学习计算机语言，从 Java ME 开始走上漫漫编程之路。但技术更新换代之快曾让白泽沮丧过，他学会 AS2 没多久，AS3 就出现了，对于不是计算机科班出身的他来说，学习过程是非常痛苦的。白泽每天给自己制定一个目标，不断研究小游戏的写法、算法，甚至还写剧情。白泽在学生时代就通过朋友介绍开始做小游戏，连连看、对对碰、塔防游戏都做过，最火的一款小游戏挂广告一个月可以赚到 8000 元。这个数目在当时已经不是一个小数目了，这无疑再一次加深了白泽的游戏梦。

随着自己对技术的了解，白泽发现浮于表明的东西并不能满足自己的需求，从而他开始关注底层技术，当实现游戏功能时需要用到新技术，他就会针对性的去学习。回顾学习开发过程，白泽认为，学习过 Flash 是其优势所在，Flash 教给他的不仅仅是门技术，更重要的是一种思想。后续，白泽还从事过页游、SNS 等相关工作，那时他已经开始开发自己的框架并且利用四天时间做出一款飞机游戏——漫天火力。

漫天火力



边学边做 HTML5 引擎由此诞生

白泽是一个善于发现问题和解决问题的高手，他遨游 Flash 领域后，总结出自己一套学习方法和开发思路。这是属于他自己的价值，这也是每个程序员要学习和付出行动的。他十分渴望开发出一款优质的游戏，为了达成这个梦想，他探索不同领域的开发。他曾研究 Android 开发，由于 Android 开发相对成熟，部分

从事人员也固步自封，自封为技术大牛，融入这个圈子对白泽来说并不困难，但他并不喜欢这种死气沉沉的气氛，所以并未在 Android 领域下猛料。而是将精力转向了 HTML5，在这里他发现，这是一个神奇的技术。一处开发，处处受用。这种跨平台性能以及和 Flash 相似的 Canvas 功能，深深吸引了他。同时，他也看到了 HTML5 未来发展前景，他认为这种标准，势必将畅行于互联网。随着标准逐渐的明亮和清晰，将会为开发者提供更多的机遇和价值。

为此，他与小伙伴共同创办了 HTML5 游戏开发者社区，在社区中分享一些自己学习过程，以及引擎开发教程为主，这个引擎是白泽从零起步学习 HTML5，一路摸索出来的结晶。为了更好的传递经验，白泽将他的引擎源码全部开源，并在他的系列文章中分布讲解每个模块的功能开发，目的仅是为让多初学者能感受到开源气氛，希望能传染未来的程序员，让他们也倡导开源。同时，他也希望借此社区能向开发者传递游戏开发中的底层逻辑，以及开发思路。

关注不同平台的实现方法

拥有 HTML5 和 Flash 开发身份的白泽，对于两种技术之间的矛盾自有感悟，他从不同角度阐述了自己的观点。他认为 Flash 与 HTML5 最大的区别在于，Flash 相当于一件产品，由公司来控制它的发展，就如同是普通软件。HTML5 则不同，是一种互联网标准。此外，移动浏览器不兼容 Flash，只能用 HTML5 来做。在开发方面，两者有着相同的地方，HTML5 和 Flash 都可以通过打包技术在移动设备上生成 App。

但 AS 和 HTML5 两者之间存在着误解，AS 开发者认为 H5 开发者是来抢饭碗，H5 开发者则认为 Flash 不支持浏览器，又存在封装，是一种插件的形式，使用不方便，做出的游戏质量不高。但白泽指出，作为游戏程序员，不应过多地关注语言，而应关注不同平台的实现方法。世界上的语言有上千种，学习当下很火的语言，成为大师之后，可能这语言已经过时了。白泽推崇从底层的原理核心出发，之后就会发现很多东西时融会贯通的，在任何语言平台上，都能找到它们的影子。

移动页游新契机：基于浏览器的缓存游戏

在交谈的过程中，白泽时刻都在谈论着与游戏相关内容，随着深入探讨，白泽对记者表达了自己对移动页游的预测与机遇，他表示如今的流量问题是限制移动网页游戏发展的因素之一，突破流量限制的局限才能更好地发展移动网页游戏。

现阶段比较好的一个例子是多泡网，它的模式相当于一个商店，选择一款游戏就可以直接去玩，这是基于网络的一种未来趋势，移动页游未来方将遵循三点：快速、跨平台和数据唯一性。



PC 端登陆多泡游戏网站截图

拥抱开源 贵在坚持

谈到开源，不得不说中国的开源社区，相对国外来说发展较慢，这主要是国人的竞争心态造成的。优秀的技术或优化方案意味着产品利益的提升，而开源则可能造成利益的损失。白泽并不赞同推崇业界大牛这一行为。他认为写技术教程、公开代码都是在分享，帮助技术新人少走弯路，提高学习效率和成长速度。大牛与否，关键在于心态，不推崇大牛，自己也不去做大牛，大家是相互分享、相互学习的交流模式。



在交流中，记者深刻体会到白泽本人的一个开放心态，他的言语中无一不在透露出开放共赢的理念。他深信，技术的发展离不开开源，一味的闭门造车，只会与社区和时代脱轨，技术也难以突破瓶颈。只要将心态调整，接受开源，尽可能与更多人分享自己的知识，从他人手中获取反馈，或共同来完善和研究技术才

是一条捷径。

学习 HTML5 游戏开发的四大法则

针对入手简单的 HTML5 来说，学习并不是一件难事，而难于坚持。在学习道路上，碰到挫折和难题乃是常事，应该从多个角度和高度来审视问题。鉴于有一定开发经验，白泽总结了几条 HTML5 游戏开发建议。

模糊程序和美术、引擎和使用的界限。

做游戏尽量的不要去关注某个语言，而是关注最根本的实现原理和呈现方式，这可以极大地降低学习成本。

先找一个目标，再有针对性地去学习完成此目标的技术。在这一过程中，需要用到什么东西就去学什么东西。

作为一个程序员，要有自己的代码库，保存下来方便以后调用。

这就是白泽从一个非技术人员自学多语言的一种途径和方法，虽然方法并非适用于每个人，但这种开发思路值得让人深思。语言只是工具，而思路才能主导成功。实践是证实真知的唯一途径，不要一味的去想，而是要踏出第一步，步步为营的去学习。

原文链接：http://html5.9tech.cn/news/2014/0212/39813.html

我的北漂在路上--------时不时的停下脚步思考

终于到周末可以有时间更新自己的博客了。果不其然，一开始上班，时间就比较紧凑，每天发生的一切感觉都是在那么短短的几小时之内就过去了。在公司有时候会感觉自己收获很多，有时候也会因为自己用几个小时解决了别人一句话就可以解决问题而懊恼。不管怎么样自己的生活就在这样日复一日的过去，直到自己突然发现每天的自己都没有对自己生活有过思考或者是反思。

自己的生活习惯是不管睡的多晚，进被窝后都会用手机到《one》（韩寒监制

的一款应用，上面的文章可读性很高，推荐给大家）上去读一篇文章再睡。 前几天自己在上面读到一篇叫《日落天通苑》（点击文章名可以阅读）的文章感触很深。其实描写北漂生活的文章在网络上并不少见，其中或者励志，或者悲惨，或者苦闷等等什么都有。但是这篇文章的作者以一种不一样的经历，让我对北漂又有了深刻的认识，原来北京还有一个北漂聚集的地方，那里叫天通苑。

天通苑到中观村说远也不算太远。坐地铁倒两站就可以到。文章中对租户收租的描写，对各色人物的描写，让自己觉得生活的中关村和天通苑是两个世界。

一，坚持写博客的人都有一颗执着的心

过年公司放了七天假，自己是坚持到 30 号才离开的公司。放假期间自己把以前学习笔记整理成了博客文章。有这个想法也是因为自己在一次翻看笔记本的时候，发现笔记本已经快快被记得满满一本了，有些知识虽然当初记了笔记，但是还是忘记或者记不清了。所以就下定决心要把这些知识点通过整理成博客文章的方式温习一遍。在检验自己学习成果的同时把这些知识分享出去。

放假期间总共整理了七篇关于 C#的系列文章，名字叫《C#小知识》。目前还有两到三篇没有整理出来（主要是没时间）。有时候人需要耐下心做一件事，这样才会有所收获。年假自己给自己的目标就是把这些学习笔记整理成博客文章。在发表到博客园，CSDN 和 51CTO 后，效果还是不错的，有些细心的读者指出了其中的错误，自己及时的改正了这些错误，同时也学习了更多的知识。年后还成功在 CSDN 为这个系列申请了专栏。还有一件令自己非常高兴的事，前天收到 51CTO 的站内消息，说自己关于 13 年年终总结的文章获得了文章优秀奖，会收到一个大的阿狸抱枕的礼物。这也算是自己写博客路上的一点鼓励吧。

为什么自己一直坚持写博客？记不清自己为了写一篇博客，想示例代码，写示例代码，中间不可预期的又发现新的问题，解决这些新问题，度过了多少个不眠夜。有时候也想过干嘛这么累的一直坚持写博客呢，又没有人给自己稿费，不定时的还会收到讥讽的评论。现在想想理由也特简单，用博客记录进步的点滴，自己的目标不管有多远，只要自己一步一步的走，就算达不到目的，也不会有什么遗憾。因为自己的这些文章见证了自己曾经努力过。

对于不写博客的人，不可能切身体会到写博客的辛苦，特别是写一篇优秀的博客。有时候，一篇优秀的文章需要作者付出的不是一天或者两天的时间。特别

对于技术文章来说，作者为了论证自己理论的正确，需要列举有代表性的例子来给读者看，这些都会花费很多时间。还有就是在写文章的时候，为了某个细节能够让读者看明白，进行深入解释的时候，不确定的又会发现新问题。

对于一篇文章，会有人给出客观的评价，指出你的缺点，让你有机会进步。有时候也会受到讥讽的评论，说这么简单的问题还拿来写博客，一个**命令就可以解决，是来炫耀么？说实话，收到这样的评论的时候，自己的心里是不好受的。但是想想那又有什么关系呢？走自己的路让别人说去吧，反正我也没有求着你来看我写的博客。

对于概念性问题，每个人的理解程度是不一样的，对于同一个问题，不同人的解决方法也是不一样的。既然您觉我对概念的解读有误差，解决问题的办法太繁琐，为什么不指出来大家交流一下呢，为什么非要说一些讥讽的话呢？

二，关于学习方法

总体来说自己这次整理的学习笔记收到读者的反馈效果还是挺不错的。有一些人让我介绍自己的学习方法。问我有几年工作经验了。我只能说平时要做一个有心人，多留心，用心。对于自己陌生不熟悉的知识多记笔记，时间长了，自己的笔记本会见证自己的进步。

我一直觉的看书是学习的最快途径。特别是阅读经典图书，同时又能锻炼人的耐性。对于初学者刚开始看书可能会有很多不懂，但是一定要坚持弄懂，搞明白，多写一些 Demo 来练习。要不然刚开始一知半解，后来学习更深的知识就更加一头雾水了。会造成恶性循环。但是一味的看书只是在被动的学习，只有通过实践才可以出真知，才会有原来如此的感觉。

就好比我觉自己的 c#语言已经算是熟练程度了，但是在实际的工作中还是会遇到很多问题。自己现在的体会是语言在特定环境或者业务背景下的使用是变化多端的。就跟令狐冲的独孤九剑一样，剑法总共只有九招，但是在实战中却可以千变万化。

所以熟练了编程语言，万里长征才开始第一步。

三，事情真的那么简单么？

在学校的时候，自己身边充斥着太多说这简单说那简单的人，原因是他们自己根本不会去做。对于有一定经验的人都知道，除非你对所要解决的问题特别的

熟练，有些问题想着很简单，但是在解决它过程中总会遇到新的问题。作为程序员，我觉得应该把“这很简单。。。”这句话从大脑里面根本剔除。要不然这只能显示自己的无知。

进公司时候后，耳边几乎没有听到过什么人说过这样的话。除了总监大人！我觉的总监大人说这话也是有理由的，它要让你对所要解决的问题有信心。因为公司招人不是让你来说一句这很简单就算完成任务了，简单不简单老板不关心，老板只关心你什么时候能把问题解决了。

自己前几天就犯了这样的一个错误。在团队里我负责后端开发，前端代码经验很少（看得懂的水平）。过年后，因为做前端的孩子来的晚了一些，我就主动把他的工作给揽了下来。对于一个没有多少前端代码编写经验的我，现在很是痛苦，写代码很多时候是在尝试，有了错误也不知道从哪里下手解决它们，所以进度非常很慢。更糟糕的是，自己还给老大承诺，要在**时间完成。现在是那个后悔啊！

所以，在自己对某件事没有太多把握的情况下，不要认为那是多么简单的一件事。要不然那只能显示自己的无知。

四，新手该怎么写代码

曾经在微博上看到一张程序员实际写代码的时间分配图，原图找不到了，自己就大概画了一个，如下：



程序员真正写代码的时间

我觉得这非常符合我写代码的时候的状态。大部分的时间都耗费在了瞎搞上面。

我现在总结出的写代码的经验是，首先要根据自己要实现的场景把逻辑理清楚，最好先大概的画好一个流程图，，跟周围的讨论一下（或者找老大征求意见），确定自己的思想没有问题。时间充裕的话，就把它们制作成思维导图，

这样也便于自己理解。然后再开始写代码。这样写代码所花费的时间是事半功倍的，而且其中也会收获很多。

五，做一个知道付出的人

“当感觉自己的能力距自己的理想太遥远时，说明你该努力了。”生活中有太多人只看到了别人取得的好成绩，应聘到的好工作这些表面现象。谁也不会关心他们背后为之付出的努力。

有梦想并不能代表你拥有什么，值得你去自豪。但是也不代表距离太遥远而显得多么可怕。如果想有所成就，你必须做一个知道努力的人。有很多碌碌无为的人觉得自己需要一个机会，我以前也这么想，只要给自己一个机会，自己绝对可以比现在活的更好。现在自己明白了，机会是眷顾给有准备的人的。没有机会需要努力提高自己的能力，有机会更要加倍努力。因为你要对得起给自己机会的人，他本可以把机会给别人，但是他却给了你。

而且永远不要觉得自己是最努力的，不管如何总有比你更加努力的人存在。来北京之后我感觉自己每天休息的已经够晚了（这里不是比拼谁休息的晚），直到有天听同事说，总监和其它两个组的老大，每天加班到凌晨都是很正的事情。可能他们是领导我们是“小弟”，属于两种不同的人。但是目的都是一样的，领导需要为自己负责的工作努力，“小弟”需要为提高自己的能力努力。

六，我也谈“过劳死”

前几天看到一篇文章叫《复旦女博士的生前反省：为什么是我得癌症？》（点击名字看原文），感触很深。其中我体会到的很多意思都是说，人应该活得快乐一点不能太拼，为了事业牺牲自己的健康。

我不否定她在人生最后的日子对人生的解读。试想把我们自己放到“人生快到尽头”的环境下，我们都会回顾自己的人生。如果有机会也会告诫别人一些自己觉得曾经做错了的事情。

人生没有办法，人活着就是为了活得更好，能够有所成就。如果有一天发现身边有无数人比自己优秀。如果自己还是以一副过的平平淡淡就好的心态来要求自己，注定自己也会一生默默无闻，当然了高富帅，富二代除外。

网络上有太多这样的文章，阅读的过程中自己的同情心也确实会让自己感动甚至流泪。但是我觉的看问题应该辩证的看。就好比网络上有很多励志文章，主

人公都是通过付出**的艰辛取得成功，看过后立马觉得自己如果这么做也会成功。自己也这么做过，直到突然有天发现，任何人的成功都有其独特的一面，自己不可能去效仿别人，复制别人的成功。

时间不会为了任何一个人而停顿，就算自己真的过劳死，相信也不会有太多的遗憾，因为自己并没有虚度光阴。

七，工场的一些体会

李开复作为大学生心灵导师，影响了 80 后甚至现在的 90 后。所以创新工场在很多大学生眼中，还是比较有影响力的。有些人文自己工场的招聘条件高么，是不是需要在某些方面有特别突出的特长。

曾将创新工场在自己心中确实是一个神圣的地方。但是当自己身处时，发现也没什么。对于我所接受的招聘和我所理解的工场招聘就是，需要一定的经验和优秀的解决问题和自我学习的能力。工场里孵化的都是创业团队，项目工期比较紧凑，就算是周六也会发现有很多团队还在加班。所以，团队不可能招聘一个没有任何经验的人，然后花费大流量时间去培训你。所以，只要你想进工场体会，只要自己在某些方面有一定的经验和能力，还是很容易进的。

在工场确实可以学到自己去大公司和小公司都接触不到的东西（博主在小公司和大公司都实习过）。技术就不用说了，接触新的项目肯定有很多技术和软件设计思想可以学习。其它的包括创业中遇到的各种问题，带领团队的问题，这些都能够让自己有机会接触得到。所以在工场，不尽可以学习得到新知识，也会有机会去修身，去学习创业中遇到问题该怎么解决，带领一个团队需要什么素养。

创业是很辛苦的，以前只是在电视上或者书本上看到，很多事情都是在酒桌上谈成的。现在自己是深有体会，也不要想着老板们都愿意这样去干，但是为了团队，没有办法。其实我觉得这才是中国式最大潜规则。

前天吃饭我觉得我犯了两个错误。因为工场是大锅饭，平时我盛完饭都是找个地方一个人吃。那天老板刚好路过我身边看到我在一个人吃饭，就跟我说他也在那里吃了，我就说了声好。更不可思议的是在整个吃饭过程中我就没跟他说一句话。

还有一个细节做的不好，就是在吃饭的过程中我去拿了一张餐巾纸。事实上不管老板需要不需要我都应该给老板也带一张的。直到老板自己去拿餐巾纸的时

候，我才发现了这个生活细节。如果抓住这些细节，本可以给自己的个人素质加很多分的，但是自己却没有发现。

到最后自己也才发现，也许只有在创新工场自己才有机会，让老板跟自己坐到一起吃饭。所以生活中应该注意细节。

八，我佩服的三个人

在我眼中做技术的有两种人：一种叫大牛，一种叫大神。

一个人如果技术牛 X 可以被称为大牛，但是你要被称为大神，不仅要技术牛 X 更要人品 OK。自己佩服的大神有三个人：

第一个，“左耳朵耗子”—陈浩。怎么说呢，可能大神们对于自己的世界都是比较孤傲的吧。他的技术博客《酷壳》是自己收获最多的博客之一。

第二个，“圣殿骑士” 。一个用技术游历各国。不紧让自己的人生有意义，而且很精彩。

第三个，王垠。怎么说呢，有网友评论他是“狂人”。看王垠的文章，确实是，他思考问题的方式确实跟别人不一样。而且每次读他写的文章自己都会收获很多。

尾声：

好吧又到结尾了，用一位同事空间日志里面的话，作为结尾吧。“青春的尾声悄然敲响，我站在尽头送别剩下为数不多的光阴。婚礼上同学聚会上，越来越多的人开始为人妻为人母，我们在经受着一场不可逆的转变。转变以前是快乐的，转变后也会是另一种快乐，但是转变的过程是痛苦的，面对终将失去的，我们有一万个舍不得却又不得不接受。”

原文链接：http://blog.csdn.net/yisuowushinian/article/details/19244625

纸书的秘密

在这本书付印的前一天，本书的责任编辑杨海玲老师找到了我说：书要出版了，你要不要写个前言和读者说两句？我说：来得及吗？杨老师郑重地看着我点了点头。好吧，那就说两句，于是就有了这一篇前言。

我小学四年级之前一直在农村居住，那时候没见过汽车，只见过马车和小推车。搬到城市之后才发现，这世上居然有汽车这种交通工具，那时我的理想是当一名 212 吉普车的司机。后来见的车多了，觉得司机不够神气，我希望拥有一辆自己的汽车。这个理想在长大后也实现了。但是我从没有想过自己会出版一本书！今天这本书也要出版了，我很欣慰。

现在回想起来，这本书的诞生经历是传奇和值得称道的。我从来没有出版过一本书，曾经有几次心血来潮想写本技术书，结果规划完目录后自己已经疲惫不堪了，看着层峦叠嶂的目录结构，我仿佛走在一个没有出口的迷宫，想到以后的每个日子都要用业余时间去写这些枯燥的技术文字，我仿佛置身凄风苦雨中而茫然若失，所以，写书计划还没开始，就已经结束了。但是大家都知道，胸中有丘壑的人总能找到吐槽的土壤，是金子总会发光。2012 年 8 月微信推出了公众平台，我终于找到了一个写作的契机。

2012 年 12 月，我开始运营自己的微信公众平台“MacTalk By 池建强”，几乎每天一篇，半年内写了 100 多篇文章。之所以能够坚持下来，我想得益于公众平台的交互性和正向激励，如果每天都有人告诉你，晚上不读完你推送的文章就不睡觉，你也会坚持下去的。慢慢地读者越来越多，很多数字出版界的朋友也加入了这个队伍。突然有一天，他们不再潜伏，而是爬起来告诉我，你的这些文字该收拾收拾出本书了！我原本是不信的，不过，先是多看的朋友这么说，然后是豆瓣，然后是亚马逊，然后我就信了。这种感觉就像是我一个人混迹于敌占区多年，突然冒出了很多自己的同志，拍着我的肩膀说：小同志，你干得不错！我感到无比荣耀，紧紧握住他们的双手，激动的说：可算找到组织了，其实，俺早就想出本书！

于是电子版的《MacTalk·人生元编程》问世了，书中的内容截止在 2013 年 8 月底。

电子书上线后获得的赞誉超乎我的想象，在多看平台上线当天就跃居畅销榜第一名，从此一路榜首，整整霸占了一个月之久。随后上架的很多新书都对我这本书发起了猛烈的攻击，最终都如潮水般褪去，它们失败了，包括韩寒推出的《一个》。因为这个原因，程序员老赵时不时调侃我为“超越韩寒的男子”，虽然这句话看起来错误百出、逻辑混乱，完全不像出自一个优秀程序员之口，但我听起来还是若有所思的。

截止到 2013 年底，多看上的《MacTalk·人生元编程》获得了 700 多个评分，平均得分 9.1，另有 230 多条读者评论。有些读者说这是他们的第一本电子书，有些读者说一口气读完，酣畅淋漓。我的感受有两点：

我为电子书事业做出了些许贡献；

又不是武侠小说，读那么快有必要吗？

事情终于到了峰回路转的时候。电子书虽然风光，但仅限于电子阅读的领域，而且，单纯的电子书是没有 ISBN 的，想做个图书认证都不行，还不能签售，你总不能卖本书搭个镌刻版 Kindle 吧。这时候另一群关键人物出现了，他们是传统出版界的好朋友。每个人都告诉我，电子阅读确实是未来的方向，你已经考虑了未来的事儿，那现在的事要不要管？只有电子版，你让广大钟情于纸书的群众怎么看？让想要签名版的读者怎么看？……于是我觉得，是时候出一本升级版的《MacTalk·人生元编程》了。最终这个重担落到了程序员的好朋友、人民邮电出版社的杨海玲老师肩上。

和人民邮电出版社的合作是轻松愉快的。我继续写作，除了完善之前的文字之外，还要为这本纸质版再增加十万字。出版社的编辑和美编则开始对于原有的文字进行校对和美术设计。一切都很顺利，除了我偶尔出现写不下去或改不下去的情形。杨老师对我是非常宽容的，遇到这种情况，她就会语重心长地告诉我：对你我心里有底，选题咱已经报去了，选题号也已经批了，情况就是这么个情况，所以您写也得写，不写也得写，还是写吧。于是我的文章顺利地完成了……

剩下的就是杨老师和编辑们的工作了，他们像打磨璞玉一样一点一点地打磨这本书，从文字、内容、印张和色系选择，到最终的版式设计和封面设计，一个细节都不放过。修改顺利的时候他们不会来找我，偶尔找我的时候也是咨询一下我的个人意见，我只要一言不发或若有所思就可以了，因为杨老师他们总能找到

最佳的做法。我唯一的任务就是等待，或想点儿别的。

最终的成书就是您手中的这本书，它的封面设计和正文版式都是独一无二的。书封简约大方，干净得一塌糊涂，体现了 MacTalk 所倡导的纯粹之美。没有腰封，没有邀请业界大牛和土豪在封底写推荐语（年底了，他们都在做新年计划，我就甭给人家添麻烦了）。正文采用图文并茂的方式，页码和标题都精心设计过，如果您获得了良好的阅读体验，那么我们要感谢这本书的美术设计。

书的内容比原来的电子版增加了 Mac 入门和演化史系列、Linux 系列、Vim 插件系列、职场系列、编程与写作等内容，全书共计 70 多篇文章，划分为六大主题：Mac、程序员与编程、科技与人文、 人物、工具、职场。篇篇独立成文，可拆可合，您可以随时翻阅。Mac 主题的最后部分是 130 个 Mac 使用技巧，供 Mac 用户参考使用。另外，我根据书中的主题和时间线，重新调整了文章的顺序，更符合读者阅读习惯。

这既不是传统意义上的 Mac 用户手册或技术图书，也不是一本思想文集，而是一个 70 后程序员，以柔软的笔触，立于 Mac 之上，讲述技术与人文的故事，起于 Mac 而不止 Mac。书中写了 Mac，说了苹果，聊了技术，侃了人生元编程，汇聚了我个人的经历和悲喜。您读完了，有收获就是好事，不用想的太多。如果书里在讲 Mac，那就是苹果公司的 Mac；如果在说 Shell，那就是操作系统的 Shell，如果在聊程序员，那就是你我身边的程序员。我尽量做到文字简单、内容有趣，因为简单和有趣也是一种力量，希望您读完这本书以后，也能拥有这种力量。

王小波说，“每一本书都应该有趣，对于一些书来说，有趣是它存在的理由；对于另外一些书，有趣是它应该达到的标准”。我希望这本书达到了有趣的标准，同时还讲了一些有用的东西。如果您说没达到，那我下次继续努力。

在写作的过程中，我得到了很多朋友的帮助，包括技术圈和出版界的朋友。其中冯大辉先生在我运营微信平台和写作的过程中给予了诸多支持和推荐，并为此书作序；微信平台“二爷鉴书”的作者邱岳为此书作序并多次推荐；网友“密码有误”为此书作序并进行了部分文字的校对。杨海玲老师在成书的过程中起到了中流砥柱的作用，没有她的催促和辛劳的工作，此书无法出版，在此一并谢过。

特别感谢我的妻子和女儿，写作占用了我大量的业余时间，虽然我很努力地

抽出时间去陪伴家人，但在这方面显然是不达标的。没有她们的包容、照顾和期盼，此书无法出版。

最后，我把这本书献给我的父母。我的父母就像千千万万的普通父母一样，永远在远方默默注视着子女的成长。我在他们身边得到了最好的教育和关怀，他们是这个世界上我最敬爱的人！

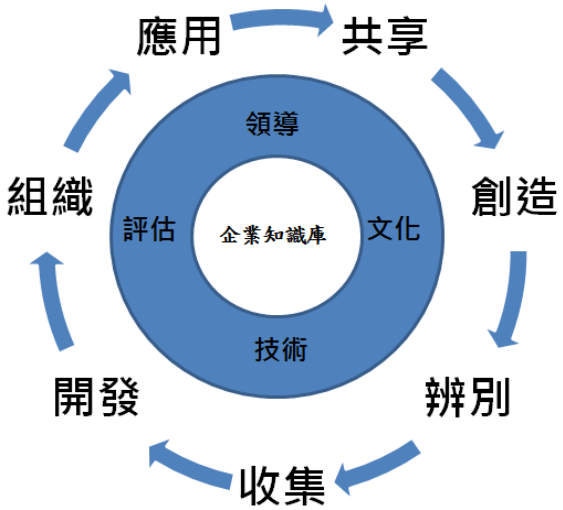
原文链接：http://macshuo.com/

一个技术人的知识管理方法论

知识管理方法论解决的是效率和效果的问题，其实没有方法论的时候，足够的专注和努力也是可以的，许三多就是这方面的典型代表,他几乎没有什么技巧性 的方法，完全是靠执着和努力，这种”傻气”有一个中肯的名字叫做”钝感力”。方法论并不意味着能够”空手套白狼”，其出发点并不是投机取巧的心理，而是一种自我调整过程中摸索出来一套最优方案组合；这样看个人知识管理方法论有两个特点:

[1] 不是那么具有普适性,至少不是任何人都能全盘拿过来用

[2] 方法论是动态的,会根据个人的情况进化



个人知识管理方法论的关键在于其有效性,所以在不断的进化过程 中，一些无效的行为会识别出来进行剪除；比如,我从来不用浏览器收藏，也不会囤积电子书，因为我发现收藏过的文章几乎都不会再看，囤积的电子书就那么一两 本会看；对我来说这两件事情用下面的方式是有效的：在 Evernote 里面有一个”待读”列表,我会逐渐把里面的文章消化掉;电子书要读的,我会放在 Kindle 里面.下面这张图就是我知识管理方法论的概况:



好多人宁可先陷入各种细节中也不愿意去想想自己目标是什么,回避最开始的目标设定环节.翻翻讲时间管理的书,很多时间管理技法的书都是从如何操作开始,而关键的并不是技法,是目标,试想如果没有目标,或者没有想清楚目标,那么使用各种技巧争取来的整块时间用来做什么呢?

类似的，知识管理也是要从目标设定开始，先确定做事的意义所在，做到什么程度；否则你投入了很多精力在一个领域，你得到你想要的了吗？你达到预期的效果了吗？没法回答！”但行好事，莫问前程”，”本想顺路打劫，哪想弄假成真”这两种心态说不上坏，但也算不上好;

设定了目标，就有了目的地，从你现在的位置到目的地怎么完成这段旅程？这就是路线图;

我们读书的时候,有目录，目录其实就是一本书的路线图；阅读过程中善用目录是学问的，详情请参见<如何阅读一本书>.我们需要拟定一个类似的 东西，路线图的好处是，我们在任一时刻都知道自己在什么位置，下一步要到哪里,障碍是什么.有什么方法可以帮我们克服障碍.

没有目的地的船是无所谓偏离航道的.

输入

输入有很多方式，或者说有太多的方式，在信息轰炸情况下，获取遴选优质的信息是必备技能;RSS,邮件推送是我最常用的信息获取方式;到目前为止我 还是排斥微博，微信这类信息渠道，我发现只要是重要的信息，不上微博也会有其

它渠道看到,而微博获取信息的成本太高了，超级碎片化无关联的信息，很难集中 精力，所以早早的玩了一阵微博后，就决然退出不玩了；微信也类似,社交网络在信息获取上的特点是你关注了什么人就得到了什么信息,明白了这一点,完全不必 拘泥于微博,可以通过 RSS 之类完成这个过程；对于豆瓣我有两种截然不同的用途，一方面维护了一个资讯小站，另一方面我关注了一堆和电影，音 乐相关的人；因为豆瓣小站和个人主页的时间线有明显产品形式上的区分，所以很容易完成两种状态的切换。还有一点要强调的是:作为一个成熟的人，一定要意识到你在社交媒体上的种种行为对别人来讲是没有多大意义的;

输入过水

读技术类的书的时候，我都会预测一下，这本书要是我写，我会怎样注水凑字数；开始阅读的时候，就会比较有意识的把注水的部分略过，比如：讲 Linux 的书注水的方式就是大篇幅讲如何安装，没节操点就把安装输出也打印出来，然后罗列一堆命令,逐一解释；讲大数据的书，就先大篇幅讲大数据发展现状，前景，展 望… 在图书信息组织上，这种信息无可厚非，逻辑上是说的通的，但，这是你想从书中得到的吗?

高强度的输入过程

不得不承认的是，在输入的过程中，信息会出现丢失；所以，会有一些高强度的方法来尽可能的减少信息输入过程中的丢失,比如开源项目代码注释版，手敲代 码，翻译文章，为代码绘制流程图等等；这些都不轻松,比如翻译文章可以试一下，读一篇技术文章和信达雅的翻译，投入的精力可能差很多。很多人无法理解技术 人的这种行为，认为难以理解，我是这样解释的：”作为一个技术人，发现一个好的项目，恨不能把自己也编译到代码里面去”.

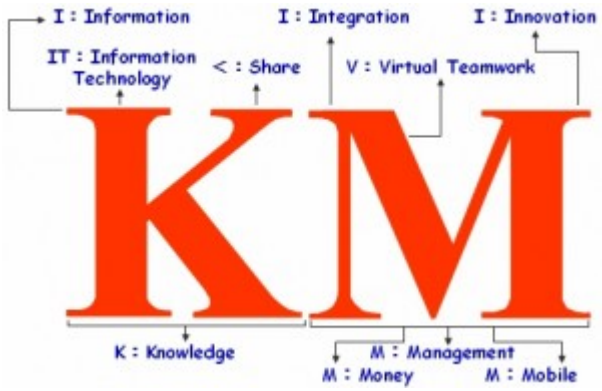
榨取每一次输入的价值

我把每一次的输入都当做”线索”；买到一本好书的时候我会把这个作者其它的作品也看一下；搜索到一个资料的时候，我会关注一下这个网站，如果是平台就注册 一下；如果是 Blog 就订阅一下 RSS，就看看这个博客的主人还关注了哪些人；如果是中文作者，看看他们有没有豆瓣 ID，你认可的人读过的书，可能是一个 非常好的阅读线索。

自己的错误是最好的输入

我曾经开玩笑说”小时候把错误较多试卷盖住的孩子，长大后会习惯性Catch 所有异常而不处理”；自己犯的错误是最佳的输入,这要么是知识盲点，要么是思 维缺陷，相比那些已经做得很好的事情，这些错误才真正包含让你更强的信息量。面对错误的那种挫败感和后悔情绪是于事无补的，无任何建设性，”往者不可谏，来者犹可追”说的就是这个道理。我第一次看到”破甑不顾”这个成语真心佩服:

《后汉书·郭泰传》：“客居太原，荷甑堕地，不顾而去。林宗见而问其意，对曰：‘甑已破矣，视之何益。’



内化

内化，就是有自己的看法，自己的理解；”自己的看法”，这个东西在咱们这里挺稀罕，我们被灌输了太多标准化的东西，自己的看法被大大的弱化了；内化就是用 自己的思维方式话语方式去描述一个东西，一个比较好玩的例子就是王朔用北京话翻译了一遍<金刚经>和<六祖坛经>.

内化不是简单记住，知识需要/值得记住的部分是有限的，把可以 RTFM 从大脑移除，让值得思考的部分留在大脑;

一本书或者一份资料放在面前，没有必要完全按照现有的结构去接受它，可以进行信息的结构上的重构；这种重构只有一个标准：自己理解起来方便.

多种视角

从多个视角来观察事物，可以获得更深的印象更好的理解，如果只有一种视角来观察事物，那么看待事物的肯定是片面的；学习同样一种知识也需要多个角度，如果只是一种角度，那最后脱离了这种视角就无法解决问题了；比如，我们熟记 TCP/IP 相关的各种资料，却不知道在现实中这些参数对应的问题和现象是什么，那 么这就是典型的一种维度缺失,只适合用来考试而不是自己掌握的技能;

关系型数据库实际上是对现实世界的一种抽象方式,不同的抽象方式落实在设计中就是不同的数据表,不同的数据字段; 当我们接触一个新知识的时候,书本上给我们只是知识的一个切片,一种抽象而已;避免盲人摸象的尴尬就需要从多个视角去认识同一个事物.

多种视角实际上是对书本知识的突破,因为书本上通常只提供一两种视角的解读，从多个视角去分析解构知识,能够获得”自己的看法”。换句话说，在从多个视 角去理解一个知识的时候,实际上完成了知识的内化过程.如果只用一种方式了解一个事物,那你就没有真正理解它。在我们所受的教育中，一个非常的变态的概念 就是”标准答案”，一个更变态的是”标准过程”，前者还只是答案正确即可，后者还要要求过程也正确。横切苹果的案例实际上就是一个事物从不同视角切入时的 奇妙景象。

高中时代有一个事情，一个运算能力超强的同学，在解决物理试卷上往对面扔沙袋，扔多少个对面小车会停住的问题时，没有使用数学归纳发，而是1,2,3,4,5….这样一个个算出来的，算到第 21 个的时候，得到答案；这个事情留下印象太深了，使用数学归纳法的也未必真正理解其解法的合理性，而这位同学使用了一种自己理解且擅长的方式解出了问题,值得琢磨。

如何找到多个视角呢？这里有一个工具就是”提问”；比如我们现在遇到一个新的东西：Redis，在了解了基本的情况之后，我们可以提出一堆问题:

Redis 和 Memcached 区别是什么?本质差异在哪里?

Redis 的性能怎么样?和 Memcache 相比如何?

Redis 安装,部署,使用上有什么需要注意的?

Redis 的数据结构在实际业务场景中有哪些”最佳实践”?

Redis 在 NoSQL 体系中是怎么分类的?为什么?

Redis 的 Bitmap 有什么妙用?在以前这些问题是怎么解决的?

Redis 模拟队列和 Rabbitmq 队列

……..

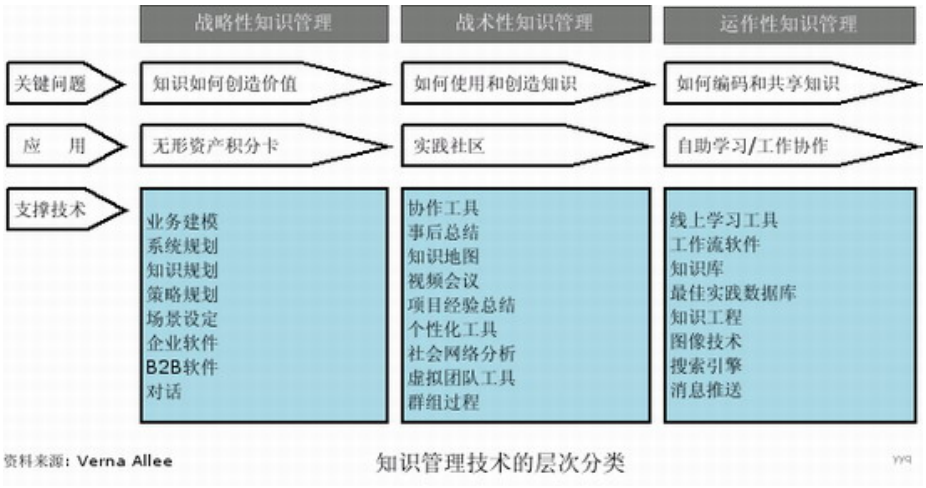
这里可以提出的问题太多了，甚至有些问题需要去代码里面寻找答案了。能够提出问题，实际上是强调了”主动性”，通过主动思考发现其中的疑问，而不是被动的 灌输；一开始通过提问打开多种视角的时候，效果可能非常差，主要是

问题问得太水了，只要多练习，特别是针对一种类型的事物反复练习再次遇到同类型的事物基 本上条件反射就能提出质量较高的问题.

可复用视角(思考模型)

同一类事物,我们提出的问题往往是类似的,比如网站系统会考虑页面缓存,静态动态资源分离,SEO 优化等等;如果是数据库方案会考虑业务数据的使用情况,数据增长情况，对事务的要求；如果是接口调用，会考虑安全问题，信任问题等等；这些对同一类事物可复用的视角，可以固化下来作为一种思考模型，省得每一次 都从头开始想起了，在不断使用过程中思考模型也不断的调整优化;

我至今难以理解的就是政治课和政治试题,这门课你不能用自己的语言去解答，需要按照书本上的标准描述去回答，因为老师要计算得分点，在快速判卷的情况下,死记硬背标准话语模式是有效的.



系统化

系统化，实际上就是建立知识点之间的联系，不再孤立，把它融入你的知识体系，说句矫情的话就是”不让一滴水蒸发最好的方法就是把它放入大海”。系统化的知识是不容易遗忘的,即使忘掉了,也可以顺藤摸瓜把相关的东西全拎出来;如何做到系统化呢？这里我常用的工具是

[1]“画图”

[2]“讲故事”

画图，有两种方式；一种是直接用纸笔画图，纸笔没有什么讲究，随便写，随便画；一种是思维导图工具，思 维导图工具用了好多年，在还没有接触过电脑的时候我就开始用思维导图，这也是让我接触电脑后最为惊喜的软件。知识长

时间不用必然存在遗忘，不必纠结，我们 能做就是留下足够的”线索”或者说是索引，需要的时候有足够的线索唤醒就 OK,图形化的内容由于直观简洁且都是按照自己的思路整理出来的，再次进入真的很 快。今年在项目中再次用到 Storm 的时候,我找到了当时的思维导图和笔记，有种热数据瞬间加载的感觉，非常棒.

讲故事，这也是常用的一个方法，王朔在<致女儿书>里面就是用讲故事的方式把人类进化史和自 己家族的历史讲了一遍,也有文章用某种语言设计过程中的趣事八卦串联起来语言的演变过程，这些都是把枯燥的知识有趣化，孤立的知识系统化的好方 法.<苏菲的世界> <上帝掷骰子吗?>都是这方面的典范;

讲故事还有一种套路就是跨学科讲故事；高中时候的生物老师是这方面的好手，他经常这样:”在茫茫的草原 上，各种动物刚刚渡过了寒冬，一场在幸存者之间的杀戮马上就要开始，同学们想一想这个食物链是怎样的?今天我们要讲的就是食物链.....” “李白斗酒诗百篇,，白酒经过消化道发生了哪些变化?”。就这样天马行空但从不喧宾夺主，这位从不用普通话讲课的老师，给同学们留下无数谈资；我们的生物课 成绩也是相当不错;

我也试过比喻类比之类的方式,发现这种路子开始的时候有效，慢慢就会把我带上歪路；理由很简单，既然是客观上两个截然不同的东西，比喻和类比只能在一定 程度上进行，分寸把握不好反而会影响理解。目前对我而言，还在尝试中的方法是 TRIZ 理论,还不好说效果如何,已经验证行之有效的方法就是画图和讲故事。

在我们所受的教育中，特别是在面对考试的时候,特别强调的就是”知识点” “重点”，零散的知识点是经不起时间考验的，好在学校里面通过反复的练习强化，来对抗这个记忆衰退。当我们走出校园，开始在工作中使用掌握的知识的时候,还有重点吗?只要影响你解决问题的都是重点.

重点？知识深度和广度

重点,是一种非常功利化的概念,是针对考试去讲的，隐含的意思是能够立竿见影有较高的 ROI。但实际工作中出现的问题不是 XX 命题小组搞出来的了，哪里有什么重点.之前和别人聊这个问题的时候，我们把知识掌握情况大致划分成下面几种:

已经掌握的知识,需要解决的问题是抗遗忘

以为自己掌握而实际未掌握的知识,一旦遇到问题就能发现,及时弥补即可

知道自己不知道的知识,很简单,意识到短板,弥补即可

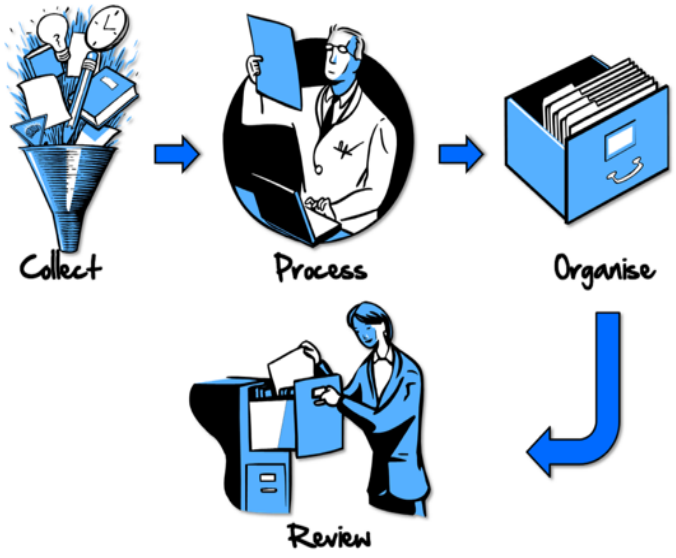
不知道自己不知道的知识,这种是最可怕的,往往会做出错误归因

特别关注一下最后一点，比如我们在解决某个技术问题时用了很久，表面原因是我们用很久才定位问题,其深层次的原因可能是在于我们对操作系统,对网络知识 掌握不扎实,不能够快速的定位或者排除某个可选项;所以我常常和朋友开玩笑说,基础知识是那种平时看没什么用，用到的时候抱不了佛脚的知识，甚至你都意识 不到是基础知识缺乏导致的问题,这里的基础知识是不限于计算机领域的.

知识点之间建立联系，从某个点看,要么是在深度上下功夫,要么是在广度上下功夫;深度层面探究原理,机制,思想;广度方面看应用,看实践;深度和广度两者是互相影响的，这对我们也是一策略调整的机会：深入研究一段时间之后可以透透气看看应用层面的东西,反之亦然.

注意止损

无论是深度探索还是广度探索，可能在某个点上,我们投入了很多精力却收获很少,ROI很低,边际效益递减;这时候,需要提醒自己适时止损,跳出;可能过一 段时间之后,再回头看的时候就会有新的理解了,可以进行下去了。其实这是一种有效避免小巷思维的方法.比如我研究一个开源项目的时候,有一个地方想了两天没明白为什么这样设计,我给自己的截止时间就是两天,然后我就把这个事情挂起了,后来看文档的时候看到了那个设计的解释,豁然开朗.



输出

掌握知识一定要有输出思维,输出方式可以是在项目中实践,可以是给自己或别人讲清楚,可以是一篇文章或者是一副图,但是无论如何一定要有输出;信息闭塞时 代,一招鲜吃遍天和教会徒弟饿死师傅的情况并存,但是网络打破了这种情况,只有分享和思辨才能保持知识的新鲜。我老家所在的村庄有很多做豆腐的,但是只有 一家从南方学来了制作”豆腐泡”的工艺,每次做的时候都是大门紧闭,对外人一字不漏;后来接触网络,我在搜索引擎中搜索的前十个问题中就有这个,原来关键 在于收水和油温;看,这种知识壁垒,在网络时代是很容易打破的;

输出最简单的形式就是讲给自己听了,你以为掌握了,开始尝试把这个东西讲出来的时候就颠三倒四,是真掌握了吗?输出给同事,既然是输出给同事,占用别人的 时间,如果你提供的信息混乱没有干货,真是罪莫大焉,出于对听众负责的态度也要保证分享内容的质量;输出为一篇博客文章,只要不是涉及商业机密的内容,总 是有好处的.

输出思维一方面强迫我们用主动的和”实用”的标准去认知,一方面通过外界反馈调整;如果没有输出,学了就是学了,连一个检验是否真的懂了的方式都没有,有什么意义呢?

原文链接：http://elevencitys.com/

ReactiveCocoa - iOS 开发的新框架

什么是 ReactiveCocoa

ReactiveCocoa（其简称为 RAC）是由 Github 开源的一个应用于 iOS 和 OS X 开发的新框架。RAC 具有函数式编程和响应式编程的特性。它主要吸取了.Net 的 Reactive Extensions 的设计和实现。

ReactiveCocoa 试图解决什么问题

经过一段时间的研究，我认为 ReactiveCocoa 试图解决以下 3 个问题：

传统 iOS 开发过程中，状态以及状态之间依赖过多的问题

传统 MVC 架构的问题：Controller 比较复杂，可测试性差

提供统一的消息传递机制

传统 iOS 开发过程中，状态以及状态之间依赖过多的问题

我们在开发 iOS 应用时，一个界面元素的状态很可能受多个其它界面元素或后台状态的影响。

例如，在用户帐户的登录界面，通常会有 2 个输入框（分别输入帐号和密码）和一个登录按钮。如果我们要加入一个限制条件：当用户输入完帐号和密码，并且登录的网络请求还未发出时，确定按钮才可以点击。通常情况下，我们需要监听这两个输入框的状态变化以及登录的网络请求状态，然后修改另一个控件的 enabled 状态。

传统的写法如下（该示例代码修改自 ReactiveCocoa 官网 ）：

```
1       static void *ObservationContext = &ObservationContext;
2       - (void)viewDidLoad {
3           [super viewDidLoad];
4           [LoginManager.sharedManager addObserver:self
5                                           forKeyPath:@"loggingIn"
6
7   options:NSKeyValueObservingOptionInitial
8
9   context:&ObservationContext];
```

```
10              [self.usernameTextField addTarget:self
11  action:@selector(updateLogInButton)
12
13  forControlEvents:UIControlEventEditingChanged];
14              [self.passwordTextField addTarget:self
15  action:@selector(updateLogInButton)
16
17  forControlEvents:UIControlEventEditingChanged];
18      }
19      - (void)updateLogInButton {
20              BOOL textFieldsNonEmpty =
21  self.usernameTextField.text.length > 0 &&
22  self.passwordTextField.text.length > 0;
23              BOOL readyToLogIn
24  = !LoginManager.sharedManager.isLoggingIn && !self.loggedIn;
25              self.logInButton.enabled = textFieldsNonEmpty &&
26  readyToLogIn;
27      }
28      - (void)observeValueForKeyPath:(NSString *)keyPath
29  ofObject:(id)object
30                                      change:(NSDictionary *)change
31  context:(void *)context {
            if (context == ObservationContext) {
                [self updateLogInButton];
            } else {
                [super observeValueForKeyPath:keyPath ofObject:object
                                                change:change
    context:context];
            }
```

```
}
```

RAC 通过引入信号（Signal）的概念，来代替传统 iOS 开发中对于控件状态变化检查的代理（delegate）模式或 target-action 模式。因为 RAC 的信号是可以组合（combine）的，所以可以轻松地构造出另一个新的信号出来，然后将按钮的 enabled 状态与新的信号绑定。如下所示：

```
RAC(self.logInButton, enabled) = [RACSignal
    combineLatest:@[
        self.usernameTextField.rac_textSignal,
        self.passwordTextField.rac_textSignal,
        RACObserve(LoginManager.sharedManager, loggingIn),
        RACObserve(self, loggedIn)
    ] reduce:^(NSString *username, NSString *password, NSNumber
*loggingIn, NSNumber *loggedIn) {
        return @(username.length > 0 && password.length > 0
&& !loggingIn.boolValue && !loggedIn.boolValue);
    }];
```

可以看到，在引入 RAC 之后，以前散落在 action-target 或 KVO 的回调函数中的判断逻辑被统一到了一起，从而使得登录按钮的 enabled 状态被更加清晰地表达了出来。

除了组合（combine）之外，RAC 的信号还支持链式（chaining）和过滤（filter），以方便将信号进行进一步处理。

试图解决 MVC 框架的问题

对于传统的 Model-View-Controller 的框架，Controller 很容易变得比较庞大和复杂。由于 Controller 承担了 Model 和 View 之间的桥梁作用，所以 Controller 常常与对应的 View 和 Model 的耦合度非常高，这同时也造成对其做单元测试非常不容易，对 iOS 工程的单元测试大多都只在一些工具类或与界面无关的逻辑类中进行。

RAC 的信号机制很容易将某一个 Model 变量的变化与界面关联，所以非常容易应用 Model-View-ViewModel 框架。通过引入 ViewModel 层，然后用 RAC 将 ViewModel 与 View 关联，View 层的变化可以直接响应 ViewModel 层的变化，这使得 Controller 变得更加简单，由于 View 不再与 Model 绑定，也增加了 View 的可重用性。

因为引入了 ViewModel 层，所以单元测试可以在 ViewModel 层进行，iOS 工程的可测试性也大大增强了。InfoQ 也曾撰文介绍过 MVVM：《MVVM 启示录》。

统一消息传递机制

iOS 开发中有着各种消息传递机制，包括 KVO、Notification、delegation、block 以及 target-action 方式。各种消息传递机制使得开发者在做具体选择时感到困惑，例如在 objc.io 上就有专门撰文（破船的翻译 ），介绍各种消息传递机制之间的差异性。

RAC 将传统的 UI 控件事件进行了封装，使得以上各种消息传递机制都可以用 RAC 来完成。示例代码如下：

```
1        // KVO
2        [RACObserve(self, username) subscribeNext:^(id x) {
3            NSLog(@"成员变量  username  被修改成了：%@", x);
4        }];
5        // target-action
6        self.button.rac_command = [[RACCommand alloc]
7    initWithSignalBlock:^RACSignal *(id input) {
8            NSLog(@"按钮被点击");
9            return [RACSignal empty];
10       }];
11       // Notification
12       [[[NSNotificationCenter defaultCenter]
13
14   rac_addObserverForName:UIKeyboardDidChangeFrameNotification
15                                  object:nil]
```

```
16              subscribeNext:^(id x) {
17                  NSLog(@"键盘 Frame 改变");
18              }
19      ];
20      // Delegate
21      [[self rac_signalForSelector:@selector(viewWillAppear:)]
22 subscribeNext:^(id x) {
23              debugLog(@"viewWillAppear 方法被调用  %@", x);
24      }];
```

RAC 的 RACSignal 类也提供了 createSignal 方法来让用户创建自定义的信号，如下代码创建了一个下载指定网站内容的信号。

```
1       -(RACSignal *)urlResults {
2               return [RACSignal createSignal:^RACDisposable
3  *(id<RACSubscriber> subscriber) {
4                   NSError *error;
5                   NSString *result = [NSString stringWithContentsOfURL:[NSURL
6  URLWithString:@"http://www.devtang.com"]
7
8  encoding:NSUTF8StringEncoding
9
10 error:&error];
11                  NSLog(@"download");
12                  if (!result) {
13                      [subscriber sendError:error];
14                  } else {
15                      [subscriber sendNext:result];
16                      [subscriber sendCompleted];
17                  }
```

```
18                  return [RACDisposable disposableWithBlock:^{
19                      NSLog(@"clean up");
                    }];
                }];
            }
```

如何使用 ReactiveCocoa

ReactiveCocoa 可以在 iOS 和 OS X 的应用开发中使用，对于 iOS 开发者，可以将 RAC 源码下载编译后，使用编译好的 libReactiveCocoa-iOS.a 文件。

开发者也可以用 CocoaPods 来设置目标工程对 ReactiveCocoa 的依赖，只需要编辑 Podfile 文件，增加如下内容即可：

```
1                   pod 'ReactiveCocoa'
```

ReactiveCocoa 的特点

RAC 在应用中大量使用了 block，由于 Objective-C 语言的内存管理是基于引用计数 的，为了避免循环引用问题，在 block 中如果要引用 self，需要使用 @weakify(self)和@strongify(self)来避免强引用。另外，在使用时应该注意 block 的嵌套层数，不恰当的滥用多层嵌套 block 可能给程序的可维护性带来灾难。

RAC 的编程方式和传统的 MVC 方式差异巨大，所以需要较长的学习时间。并且，业界内对于 RAC 并没有广泛应用，这造成可供参考的项目和教程比较欠缺。 另外，RAC 项目本身也还在快速演进当中，1.x 版本和 2.x 版本 API 改动了许多，3.0 版本也正在快速开发中，对它的使用也需要考虑后期的升级维护问题。

作为一个 iOS 开发领域的新开源框架，ReactiveCocoa 带来了函数式编程和响应式编程的思想，值得大家关注并且学习。

原文链接：http://blog.devtang.com/blog/2014/02/11/reactivecocoa-introduction/